EXHIBIT 1

ROBERT A. VAN NEST — #84065
rvannest@kvn.com
CHRISTA M. ANDERSON — #184325
canderson@kvn.com
KEKER & VAN NEST LLP
710 Sansome Street
San Francisco, CA 94111-1704
Telephone: (415) 391-5400
Facsimile: (415) 397-7188

DONALD F. ZIMMER, JR. (SBN 112279)
fzimmer@kslaw.com
CHERYL A. SABNIS (SBN 224323)
csabnis@kslaw.com
KING & SPALDING LLP
101 Second Street – Suite 2300
San Francisco, CA 94105
Telephone: (415) 318-1200
Facsimile: (415) 318-1300

Attorneys for Defendant
GOOGLE INC.

SCOTT T. WEINGAERTNER (*Pro Hac Vice*)
sweingaertner@kslaw.com
ROBERT F. PERRY
rperry@kslaw.com
BRUCE W. BABER (*Pro Hac Vice*)
bbaber@kslaw.com
KING & SPALDING LLP
1185 Avenue of the Americas
New York, NY 10036-4003
Telephone: (212) 556-2100
Facsimile: (212) 556-2222

IAN C. BALLON (SBN 141819)
ballon@gtlaw.com
VALERIE HO (SBN 200505)
hov@gtlaw.com
HEATHER MEEKER (SBN 172148)
meekerh@gtlaw.com
GREENBERG TRAURIG, LLP
1900 University Avenue
East Palo Alto, CA 94303
Telephone: (650) 328-8500
Facsimile: (650) 328-8508

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

ORACLE AMERICA, INC.

Plaintiff,

v.

GOOGLE INC.

Defendant.

Case No. 3:10-cv-03561-WHA

Honorable Judge William Alsup

OPENING EXPERT REPORT OF DR. OWEN ASTRACHAN

TABLE OF CONTENTS

I. INTRODUCTION 4

II. DOCUMENTS AND INFORMATION CONSIDERED 6

III. BRIEF SUMMARY OF MY OPINIONS 6

IV. BRIEF BACKGROUND ON JAVA 8

V. DETAILED STATEMENT OF THE BASIS FOR MY OPINIONS ON APIs 10

- A. What is an API? 15
- B. Useful Analogies 16
- C. Purposes of an API 18
- D. The Elements of an API 20
- E. The Components of a Method Declaration 23
- F. Organizing Related Methods into Packages 27
- G. The Distinction Between An API and Its Implementation 28
- H. Sun and Oracle Also have Implemented and Distributed APIs from Other Software 35
  - 1. Sun Implemented and Distributed APIs from Previous Generations of Spreadsheets as Part of StarOffice and OpenOffice.org 36
  - 2. Sun Implemented and Distributed APIs from Linux as Part of the Solaris Operating System 39
  - 3. Oracle Implemented and Distributed APIs from IBM as Part of the Oracle Database Server 44
- I. Basic Example of Java Method Usage 46
- J. The APIs at Issue are Methods of Operation 47
- K. The Java API Package Names Are Dictated By Function 51

L. The Java API Class and Method Names Are Dictated by Function 55

M. The Java Names Are the Product of Mechanical Rules 64

N. The Organization of API Elements is Dictated By Function And Does Not Reflect Creativity 66

O. The Same Analysis Applies to the Native Files 67

P. Many API Elements are Drawn from the Public Domain and Are Not Original to Java 69

Q. The APIs at Issue Are Necessary For Basic Functionality and Interoperability 73

R. The APIs at Issue Are Demanded by the Industry 74

VI. THE ANDROID PLATFORM IS NOT VIRTUALLY IDENTICAL OR SUBSTANTIALLY SIMILAR TO THE JAVA PLATFORM 77

VII. ANDROID'S DOCUMENTATION OF THE APIs AT ISSUE IS NOT VIRTUALLY IDENTICAL OR SUBSTANTIALLY SIMILAR TO ORACLE'S DOCUMENTATION 80

VIII. THE TWELVE FILES OR PORTIONS OF FILES ALLEGED BY ORACLE TO HAVE BEEN COPIED ARE QUALITATIVELY AND QUANTITATIVELY INSIGNIFICANT AND THEY ADD NO OR VERY LITTLE VALUE TO ANDROID 82

A. TimSort Files 83

B. Security Test Files 85

C. Comments in CodeSourceTest.java and CollectionCertStoreParametersTest.java 88

EXHIBIT A: OWEN ASTRACHAN CV

EXHIBIT B: DOCUMENTS AND INFORMATION REVIEWED

EXHIBIT C: EXCEL AND STAROFFICE SPREADSHEET

EXHIBIT D: LX_BRAND SYSCALL TABLE

EXHIBIT E: SOURCE CODE FOR SLOCCOUNTER.PY AND SLOCCOUNTERTOTAL.PY

I. INTRODUCTION

1. I am Professor of the Practice of Computer Science and Director of Undergraduate Studies in the Computer Science Department at Duke University. I earned my AB degree with distinction in Mathematics from Dartmouth College and MAT (Math), MS, and PhD (Computer Science) from Duke University. I teach undergraduate computer science courses using the Java, C++, and Python programming languages, and helped develop broadly-used teaching materials, including a C++ textbook and Java language programming exercises and documentation. I received a National Science Foundation CAREER award in 1997 to incorporate design patterns in undergraduate computer science curricula, an IBM Faculty Award in 2004 to support componentization in both software and curricula, and was one of two inaugural NSF CISE Distinguished Education Fellows in 2007 to revitalize computer science education using case- and problem-based learning. My research interests have been built on understanding how best to teach and learn about object-oriented programming, software design, and computer science in general; and I am now working on developing a portfolio of substantial, interdisciplinary problems that help explain how computer science is relevant to students in the social and natural sciences. My qualifications and information regarding my prior testimony are attached hereto as Exhibit A.

2. I am being compensated for my work in this litigation at the rate of $300 an hour. My compensation does not depend in any way on the outcome of this litigation.

3. I have been asked by Google to:

a. opine on whether the Java Application Programming Interface ("API") specifications from which Oracle alleges certain parts of the Android platform are derived are methods of operation;

b. opine on whether the allegedly infringing materials are driven by functional considerations, considerations of interoperability and efficiency, industry practice or demand, or drawn from public domain material;

c. opine on whether the allegedly copyrighted Oracle works relating to the Java platform are virtually identical or substantially similar to the Android platform;

d. opine on whether the allegedly copyrighted Oracle documentation relating to the Java APIs are virtually identical or substantially similar to Google's documentation for the APIs in Android; and

e. opine on whether the 12 files and/or portions of those files alleged by Oracle to include material literally copied by Google are qualitatively and/or quantitatively insignificant.

4. I understand that I may further be asked by Google to review submissions related to copyright issues from Oracle's experts, and to provide my opinions on issues raised by any such submissions.

5. I understand that I may be called upon to testify in this case regarding my opinions and analyses set forth in this report. If called upon to testify, I may use various demonstratives, including tables or drawings, to assist in presenting my testimony.

II. DOCUMENTS AND INFORMATION CONSIDERED

6. My opinions are based on my relevant knowledge and experience, as well as review of the documents and information identified in Exhibit B.

III. BRIEF SUMMARY OF MY OPINIONS

7. "Java" may refer to three very different things: the Java programming language, the Java Application Programming Interfaces (APIs), or software source code that references and implements the APIs. In this case, except for a very small number of files addressed in Section VII below (12 files out of approximately 57,000), it is my understanding that Oracle does not allege infringement of the software source code referencing the APIs. Nor, I am informed, does it allege infringement because Android is written in the Java language. I have been informed that Oracle's claim of infringement is based on Google's creation of software source code written in the Java language that references and implements Java APIs.

8. The Java language is a programming language. As with any language, it has a basic syntax and grammar that must be followed for code written in the Java language to be understood by a computer. Java's syntax includes such utilitarian features as spacing, punctuation, and the meaning of a limited number of defined words and phrases. Starting in the mid-1990s, Sun Microsystems widely promoted the use of the Java language by developers, businesses and the general public, without restriction. It is my understanding that neither Sun nor Oracle, which acquired Sun, claims that the use of the Java language to write software is infringing.

9. As explained in more detail starting at paragraph 24, an API provides programmers with a way to access the functionality of a software service. For example, most programming

languages provide a way to calculate the square root of a number. Java does this by way of an API. In order, for example, to calculate the square root of 25.0, a Java programmer includes the text sqrt(25.0) in the text of his program. The text "sqrt(a)" — where "a" is replaced by the variable or number that the programmer for which the programmer wants to calculate the square root — is the API for the square root "method."

10. As explained in more detail starting at paragraph 52, APIs are implemented by software. For example, the sqrt(a) API can be implemented in many different ways. First, there are many different mathematical algorithms for calculating square roots (much as there are many different ways to make an apple pie). Second, there are many different ways to write the programming code that implements any given algorithm (just like two people who write down the same recipe can describe the various steps using different words and sentences). However, if one sets out to implement an API, the one part that cannot change is the API itself. For example, if one wants to implement the sqrt(a) API, one cannot change the method name from "sqrt" to, say, "squareroot." As I will discuss below, referencing an API requires the use of the API's method declarations (names, data, and data types). To implement an API, one must use all these exactly as the API requires them to be used. If even a minor change is made, code that references the API will fail to operate. That said, as noted above, there are potentially different ways to write the underlying implementing code for a given API. It is my understanding that except for the 12 files discussed below in section VII, Oracle has not identified any literal code copying by Google in this case.

11. In this case, it is my understanding that Oracle contends that referencing the API, via method names, data names, and data types, is infringing, but as I discuss below in Section V.J, the APIs at issue in this case are purely functional. In addition, with respect to

Oracle's allegation of literal copying of 12 files out of 57,000 files in Android, as I discuss below in Section VIII, it is my opinion that the allegedly copied material in these files is qualitatively and quantitatively insignificant.

12. It is my opinion that the APIs at issue are methods of operation.

13. It is my further opinion that any similarity between the names of elements (such as package, class and method names) in the implementations in these APIs in the Java and Android platforms is driven by functional considerations. It is also my opinion that any similarity between the organization of elements in the implementations in these APIs in the Java and Android platforms is also driven by functional considerations.

14. It is my opinion that many of the names of elements of the Java API were drawn from usage in other languages or platforms that pre-date Java.

15. It my opinion that Google's use of these APIs is necessary for interoperability and efficiency, and/or driven by industry demand.

16. It is my opinion that the Android platform is not virtually identical or substantially similar to the allegedly copyrighted works relating to the Java platform.

17. It is my opinion that any similarities between Google's documentation of the APIs at issue and Oracle's documentation are driven by functional considerations and industry practice regarding such documentation.

IV. BRIEF BACKGROUND ON JAVA

18. Oracle uses "Java platform" to mean a variety of interchangeable and overlapping elements. These elements purportedly include the Java programming language itself, an "object-oriented" programming language that uses syntax heavily based on prior

languages such as the "C" and "C++" programming languages. Also included in these elements are a program known as a "compiler" that creates the "bytecode" in which Java programs are executed; a virtual machine that executes the bytecode; and a set of core libraries that facilitates the development of applications for the Java platform by providing basic system or language functionalities. Java is a popular programming language, and a variety of software, including internet services and mobile applications, is written in the Java language.

19. Like any high-level programming language, the Java programming language contains many rules of grammar and syntax that cannot generally be varied. For example, a statement adding two numbers can only be written in certain ways, and the language requires specific and precise key words to express such things as variable types (integers, strings, or Booleans) and more complex object types such as dates or database queries. In addition, the Java language, like many programming languages, employs key words and operators (such as plus and minus symbols) that can only be used for specific purposes and in specific ways; using them for other purposes will cause a program to fail to function correctly. As a result, much of the structure and appearance of code written in the Java programming language is dictated by these functional considerations.

20. Java's specifications, including the specifications for the language API packages at issue here, were published or made available in various forms, including in books and on the Java website, starting with the release of version 1.0 in 1996. Several revisions have been released since then, including version 1.5, which in my understanding is the most recent version at issue in this case. Note that, at times, Java version 1.5 has also been referred to as Java version 5.0. For consistency, I will refer to it as Java 1.5.

V. DETAILED STATEMENT OF THE BASIS FOR MY OPINIONS ON APIS

21. Based on my review of Oracle's responses to Google's interrogatories, I understand that Oracle is claiming that Google's implementation of the Java API specifications for the following packages infringe Oracle's copyrights. (It is my understanding that Oracle, with the exception of the 12 files discussed in Section VIII, has not identified instances of the copying of specific code.)

java.awt.font

java.beans

java.io

java.lang

java.lang.annotation

java.lang.ref

java.lang.reflect

java.math

java.net

java.nio

java.nio.channels

java.nio.channels.spi

java.nio.charset

java.nio.charset.spi

java.security

java.security.acl

java.security.cert

java.security.interfaces

java.security.spec

java.sql

java.text

java.util

java.util.jar

java.util.logging

java.util.prefs

java.util.regex

java.util.zip

javax.crypto

javax.crypto.interfaces

javax.crypto.spec

javax.net

javax.net.ssl

javax.security.auth

javax.security.auth.callback

javax.security.auth.login

javax.security.auth.x500

javax.security.cert

javax.sql

javax.xml

javax.xml.datatype

javax.xml.namespace

javax.xml.parsers

javax.xml.transform

javax.xml.transform.dom

javax.xml.transform.sax

javax.xml.transform.stream

javax.xml.validation

javax.xml.xpath

22. Based on my review of Oracle's responses to Google's interrogatories, I understand that Oracle is also basing its infringement claim on the following native code implementations of Java API classes:

java_lang_Class.c

java_lang_Object.c

java_lang_reflect_AccessibleObject.c

java_lang_reflect_Array.c

java_lang_reflect_Constructor.c

java_lang_reflect_Field.c

java_lang_reflect_Method.c

java_lang_reflect_Proxy.c

java_lang_Runtime.c

java_lang_String.c

java_lang_System.c

java_lang_Throwable.c

java_lang_VMClassLoader.c

java_lang_VMThread.c

java_security_AccessController.c

java_util_concurrent_atomic_AtomicLong.c

sun_misc_Unsafe.c

23. Based on my review of Oracle's responses to Google's interrogatories, I understand that Oracle also bases its copyright claim on code and comments in the following files that allegedly have been copied from Oracle code or comments in Oracle's source code:

Allegedly copied test files:

AclEntryImpl.java

AclImpl.java

GroupImpl.java

OwnerImpl.java

PermissionImpl.java

PrincipalImpl.java

AclEnumerator.java

PolicyNodeImpl.java

Allegedly copied comments (but not source code):

CodeSourceTest.java

CollectionCertStoreParametersTest.java

Allegedly contain copied source code:

TimSort.java

ComparableTimSort.java

A. WHAT IS AN API?

24. An Application Programming Interface (API) is "a particular set of rules and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers." Wikipedia, *Application programming interface*, http://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=437864024 (as of July 13, 2011, 00:30 GMT). An API provides a specified and documented mechanism to invoke, operate, and interact with software services. The interface itself is implemented by software, *i.e.*, by source code that is written to provide the functionality of the interface.[1]

25. APIs are used by software developers when writing software that will utilize the functionalities operated through these communications. When used alone, the term API can refer to either the set of rules and specifications, or to the software that implements the rules and specifications and therefore is operated by the communication from another program, as explained in more detail in paragraph 52. API can also refer to either a specific "element," "component," or functionality within the API, or to a collection of

---

[1] Newton's Telecom Dictionary, 25th Edition, defines API similarly as "Software that an application program uses to request and carry out lower-level services performed by the computer's ... operating system." Sun's Java glossary (*available at* http://java.sun.com/docs/glossary.html) also provides a definition for API: "The specification of how a programmer writing an application accesses the behavior and state of classes and objects." Although these definitions use different terminology, they are not materially different from, and are in fact consistent with, the definition I have presented above.

them. This report will generally use API to mean such a collection, and "API elements" or "API components" to refer to the individual mechanisms within the collection.

B. USEFUL ANALOGIES

26. APIs are similar to the interfaces that a computer user uses to operate software, like a keyboard command or button. In each case, the person seeking to use the program does something to inform the program being used that he wants a specific action to happen, and then that action happens. No deep expertise or understanding of the inner workings of the computer system is needed by the person seeking to use the program. For example, a computer user might type the "Ctl+P" key combination or click an icon that looks like a printer, and then, in the dialog box that appears, choose the file to be printed and the number of copies that should be printed. Typing "Ctl+P" or clicking the icon would invoke the underlying printing functionality, and (once the number of copies is specified) cause the software to print the document that number of times. The user does not need to have substantial understanding of the underlying printing mechanism, he just needs to learn and remember the familiar "Ctl+P," give the necessary information (*e.g.*, number of copies he wants printed), and the computer takes care of the rest.

27. Similarly, when invoking or using an API in a software application, a programmer should not need to review or understand the underlying implementation or source code for the API, as that code has already been written. Like using "Ctl+P" to print, he only needs to know the name and functionality of the API. In order to write software that prints, a programmer would read and learn their chosen operating system's API for printing, and then invoke that API from their program, telling the API critical information like what document to print and how many copies to print. Just like typing the "Ctl+P" command or clicking the printer icon, using the name of the API element in the software invokes

the underlying functionality of the API and causes printing to happen, without the programmer needing to have a deep knowledge of the particular mechanisms that allow the API to function. Among other benefits, this means that a programmer can use an API to create software that works on different printers (color, black and white, inkjet, etc.) without knowing in detail how those different printers work, as long as the underlying implementation supports them.

28. An API also can be analogized to the interface for driving a typical car. Every car has a variety of elements that are part of the overall system of communication and operation that a driver must understand and use in order to drive the car. These elements include the gear lever, the turn signal stalk, the steering wheel, and the accelerator and brake pedals. Some of these elements provide information from the driver to the car, while others provide information from the car to the driver, and others do both things at the same time. In combination, these elements form the interface to the "application" that is the car, allowing a driver to "program" the car to do their bidding by using those elements to operate the car.

29. Thus, for example, the driver of a car can make it accelerate by pressing the accelerator. The further the driver presses the accelerator, the faster the car speeds up. The accelerator can be thought of as an API for the car that makes the car go. Every car that implements this API will have an accelerator, and each of them will share in common the fact that the car speeds up faster the further the accelerator is pressed down.

30. So long as a driver understands this functionality — that the rate of speeding up a car depends on how far the accelerator is pressed down — the driver need not know *how* this happens. Similarly, if a driver understands how the other interfaces to the car's

operation function (the turn signals, steering wheel, etc.), the driver need not know how the engine, light bulbs, or transmission work.

C. PURPOSES OF AN API

31. The primary purpose of APIs is to allow one piece of software to speak to another piece of software in a clearly defined, reusable, interoperable way. This simple goal has a number of important ramifications and benefits.

32. Familiar interfaces make it simpler to use things and to use them more expertly. When using a new car, most drivers do not think about how that particular steering wheel works. For example, the wheels of the car might be turned by a rack and pinion system or a recirculating ball system. But the steering wheel itself functions the same way regardless of how the internal steering mechanism and system is designed, *i.e.*, when the driver turns the wheel to the left, the car moves in the left direction. This interface — the same, familiar steering wheel — facilitates using the car, regardless of which specific car is being used. The same thing happens in software — using (or providing) a standard API allows the users of that API (software programmers) to move between any software platforms that provide the same API, because their familiarity and existing skills in using that particular interface transition over.

33. A defined, fixed API allows different programs to substitute for each other, which gives users the ability to move from one piece of software to another. In this way, APIs enable user choice between competing software providers, and therefore help to promote competition, innovation and choice in the software market. For example, if a software platform provides a set of APIs, a subsequently created platform that implements those same APIs (for example, by writing different source code to implement those APIs) can

improve competition and reliability because developers who use the platform would already be familiar with the APIs and so would be more able to leverage their existing knowledge and complementary software. In our printing analogy, another program that provides the same commands (like "Ctl+P") will be much easier for a user to switch to. In fact, once a user is used to "Ctl+P," he will often be confused if a program uses something else to control printing.

34. APIs also help software programmers by insulating programmers from underlying complexity. This is referred to by programmers as "encapsulation." In the car example, the internal steering system can be changed specifically because the familiar steering wheel interface (the car's "API") has hidden these implementation details from the driver. This shielding and simplification is an important part of what an interface provides — most users do not need to know the details, which has in the past allowed car manufacturers to switch from old technologies to new ones without introducing a new learning curve for consumers. The concept that moving the wheel left turns the car left remains the same, and that allows consumers to rely on this familiar concept, regardless of which car they use.

35. APIs also help programmers and the industry by allowing software to be reused. This is important; new code is difficult and expensive to write and test, and so individual programmers and corporations like to reuse code as much as possible. Even if they cannot reuse an entire program (say, because the two pieces of hardware are very different, as they are between a desktop computer and a phone) they still prefer to reuse as many parts of the software as possible. APIs help make this possible by allowing the same basic functionalities to be provided and used in a replicable, but portable, way.

D. THE ELEMENTS OF AN API

36. An API typically consists of what programmers call *methods* or, equivalently, functions. These two terms are synonyms, used somewhat interchangeably depending on the programming language. Java programmers (and therefore this report) use the term "methods." Both represent the same thing — a piece of software that performs a specific function and can be reused when needed. Methods are the primary mechanism by which programmers invoke the functionalities provided by a software system.

37. In the user interface analogy, "Ctl+P" and the printer icon are methods. A user interacting with a software program might use a menu or a menu shortcut to open a file, or to save or print what has been opened, *e.g.*, in a word processing program. Often an icon of a printer or a disk can be pressed to invoke the same functionality as choosing a menu item or the menu shortcut. In all three cases — pressing the icon, choosing the menu item, or typing a keyboard shortcut — the same underlying software is invoked and causes a specific action — printing the file or saving it, for example. The functions or methods in an API are directly analogous — just as the user might click on an icon or press a sequence of keys to use a keyboard shortcut to invoke more complicated operations such as printing or saving, the program calls the function or method to invoke and control a more complicated service or feature provided by the underlying software. For example, when a Java programmer wants to get the square root of 25, his program will have to contain the following text:

```
sqrt(25.0)
```

This will cause the underlying system to do the math and tell the program that the answer is "5.0". Similarly, to get the absolute value of -25, the program must contain `abs(-`

`25)` This will cause the underlying system to do the math and tell the program that the answer is "25."

38. Related methods are often grouped together to make them easier to use, frequently into groups called (depending on the programming language) libraries or packages. In Java, these groupings are called packages. Because Java is what is known as an "object-oriented" language, related methods are themselves encapsulated in a class, and then related classes are encapsulated into a package or a sub-package. To put it a different way, the API packages include subparts or files known as classes, and within these classes are methods. As an analogy, one can think of menus (like "File," "Edit," etc.) as "packages" of menu items, which organize the menu items so that they are grouped together in reasonable groupings. For example, to print using the menu, a user needs to know that Print is under File, rather than under Edit.

39. As an example from the Java API, the java.lang package (according to Oracle's documentation) "[p]rovides classes that are fundamental to the design of the Java programming language." One of these "fundamental" classes is the "Math" class, which Oracle describes as containing "methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions." The actual methods contained with the class are listed in this chart:

Method Name	**Functionality of the Method**
abs	Returns the absolute value of the argument. (Four variants)
acos	Returns the arc cosine of the argument.

Method Name	**Functionality of the Method**
asin	Returns the arc sine of the argument.
atan	Returns the arc tangent of the argument.
atan2	Converts rectangular coordinates (two arguments) to polar coordinates.
ceil	Returns the smallest integer that is not less than the argument, *e.g.*, if the argument is 1.9, will return 2.
cos	Returns the cosine of the argument.
exp	Returns *e* raised to the power of the argument.
floor	Returns the largest integer that is not more than the argument, *e.g.*, if the argument is 1.9, will return 1.
IEEEremainder	Returns the remainder of two arguments as prescribed by the IEEE 754 standard.
log	Returns the natural logarithm of the argument.
max	Returns the greater of two arguments, *e.g.*, if the arguments are 3 and 4, will return 4. (four variants)
min	Returns the lesser of two arguments, *e.g.*, if

Method Name	**Functionality of the Method**
	the arguments are 2 and 3, will return 2. (four variants)
pow	Returns the value of the first argument raised to the power of the second argument.
random	Returns a random number between 0 and 1.
rint	Returns the closest integer to the argument.
round	Returns the closest number to the argument.
sin	Returns the sine of the argument.
sqrt	Returns the square root of the argument.
tan	Returns the tangent of the argument.
toDegrees	Returns the result of a conversion of the argument (an angle in radians) to degrees.
toRadians	Returns the result of a conversion of the argument (an angle in degrees) to radians.

E. THE COMPONENTS OF A METHOD DECLARATION

40. Every method has several important characteristics that collectively are referred to as the "method declaration." The first is simply the method's *name*. Method names describe the purpose of the method, so that a programmer can easily memorize and recognize the purpose from the method's name, and vice-versa. A simple example of this is the method

in the table above named "abs," so named because its function is to calculate the absolute value of a number. To use a method, the programmer must know the method's name. If the programmer does not know the precise name, or knows only something similar, he cannot use the method, because the software cannot guess at what the programmer meant. For example, if a Java programmer writes "squareroot(25.0)" instead of "sqrt(25.0)", this will result in an error instead of calculating the square root of 25.0.

41. The second important characteristic of essentially every method is the set of *arguments* that the method expects to receive when invoked. When the method is called, the programmer typically provides information to the method that informs the software exactly what the programmer wants to happen, just as a user must usually specify how many copies he wants printed after he clicks the print button. The information provided to the method is called an argument (or parameter), and a method is said to "accept" the permitted arguments. The ability of a method to accept an argument is what allows a general purpose method to act on specific data.

42. For example, think of the "plus" or "add" button on a calculator. This is a "general purpose" button — it can add any numbers one can type in, not just one specific set of numbers. If one thinks of the "plus" button on a calculator as a method, the numbers one asks the calculator to add (say, 2 and 2) are the parameters to the "plus" button — those parameters determine the specific outcome of the general purpose button. Similarly, the number of copies one tells the print dialog (or print method) to print is also an argument — they again tell the general purpose function ("print") a specific behavior ("print two copies.") In the "abs" function mentioned previously, there is only one argument, and that argument is simply a number, whose absolute value the program wishes to calculate.

43. These arguments or parameters must be defined when the API is first created, and are typically limited. For example, it would not make sense to ask one's printer to print "hippopotamus" number of copies of a document — that argument must be a number. In fact, the definition of a method in many languages, including Java, will indicate what "type" of argument a function will accept, such as an integer, a string, or another data type. A steering wheel, similarly, can accept arguments of left, right, or any angle in-between, but cannot accept "up" or "down." The functionality of each method constrains what parameter(s) are acceptable, and if the proper parameters are not passed to the method, any attempt to use the method will fail.

44. When a program uses a method and passes it the arguments, the method then typically returns a *result* that the program can use for other purposes. This result is the final important characteristic of the method, and is called the *return* (sometimes the *return value).* In the calculator example, where plus is the method and the arguments are 2 and 2, the return value will be 4 — 2+2 returns 4. For the abs method, which computes the absolute value of a number, when the argument is 2 or -2, the return value will be 2.

45. The purpose of the return value is to return information that can be used by the program for other purposes. For example, after one asks one's calculator to add 2 and 2, the calculator returns "4," which one can use as the first step in the next math problem one intends to solve. Similarly, the return may be a message indicating the status of a method — for example, a "print" method might return "OK" (telling the program that the printing functionality has successfully completed) or "OUT OF PAPER" (telling the program that the printing functionality has hit a snag). These status messages would in turn be handled by other methods, possibly doing something like popping up an error message, or silently concluding that all is well and allowing the user to continue with his work.

46. To summarize, each method declaration has three parts — the name, arguments, and return. As a shorthand, programmers may refer only to the name of the method, but to fully know what method they are describing, it is necessary to know all three parts of the method declaration. They can be defined succinctly as:

name: the method name, which indicates its purpose and is used by a programmer or program to call or invoke the method.

argument: the data on which the method acts. The data passed as an argument to a method is often manipulated and referred to within the method itself.

return: the result of calling the method with specific arguments. This is "returned" to the programmer.

47. The documentation for a method will combine these pieces to form a reference for programmers using the API. For example, the brief version of the documentation for the abs method is:

int abs(int a) Returns the absolute value of an int value.

While this may not be easy for a non-programmer to understand, it is quite straightforward to a programmer:

- The first part ("int") shows that the return will be an "int" (short for an "integer"; i.e., a number). This tells the programmer what type of result to expect when using the function.

- The second part ("abs") is the name of the method.

- The part in the parentheses ("int a") is the argument. Again, this uses "int" to indicate that a single integer is expected; if the abs method is given something other than an integer, such as "hippopotamus," an error will occur. (The letter "a" is a convenient name for the argument, and can be changed without affecting compatibility.)

- The first, second and third parts discussed above comprise the "method declaration." The final part is a brief explanation of what the method does.

In combination, this short statement will allow a programmer to know how to use "abs" in their program to find the absolute value of a number. I will discuss documentation in more detail in paragraph 145. This section is intended to explain how the various parts of a method fit together.

F. ORGANIZING RELATED METHODS INTO PACKAGES

48. As noted in paragraph 38, most methods in an API are organized into packages of functionalities that group related methods together. As with the method names themselves, these packages are logically organized into functional groupings and named so as to make it easy for programmers to remember and find the functionalities they need. In Java, these groupings can be packages (the highest-level grouping, typically containing many classes), sub-packages, or classes (the lowest level of grouping, typically containing a handful of related methods). (*See, e.g.* "The Java Platform: A White Paper," Douglas Kramer, May 1996, *available at* http://java.sun.com/docs/white/platform/javaplatform.doc1.html, and "Package Members" in The Java Language Specification, Third Edition, *available at*

http://java.sun.com/docs/books/jls/third_edition/html/packages.html#7.1, for discussion of packages, class, and methods.)

49. As an example of this grouping, take the "sqrt" method, which calculates the square root of a number. This method is typically grouped together in a library or package with other, related mathematical functions, such as "sin" and "cos" (short for the trigonometric functions sine and cosine). Sqrt has been grouped with other math functions since at least the Algol programming language in 1968, and is still grouped together with them in Java and other modern languages, such as the Python and Ruby languages.

50. Methods that do not have related functionalities are not typically grouped together — for example, a method that prints text to a screen would not typically be in the same class or package as "sqrt."

51. To use a particular method, the Java programmer has to know what class, and what package, the method is in. A programmer calling the square root function or method in Java, for example, needs to know that the method is in the Math class, and the name of the method is sqrt. Frequently this is expressed in shorthand by combining the two names, so that "sqrt" becomes "Math.sqrt." The programmer also must know the other key parts of the structure — specifically that the method takes one argument (a number) and returns the square root of the argument. Once the programmer knows these things the underlying functionality can then easily be invoked, allowing the programmer to focus on the more complex task of writing their own software.

G. THE DISTINCTION BETWEEN AN API AND ITS IMPLEMENTATION

52. Every API, including the Java APIs at issue in this case, exists in two forms: the method declaration of the API (comprising the elements mentioned above — name, arguments,

and return), and the implementation of the API (which includes those three elements as well as the program logic that actually performs the steps necessary to accomplish the purpose of the method). The method declaration can be combined with a brief, factual explanation in the documentation so that developers can have a reference for the API, much like a dictionary might help someone learn and reference a language. The method declaration embodies the concept of the particular API.

53. Independent of, but related to, the API's method declaration is an implementation of the API — the actual underlying source code that implements the API and allows the API to function. An implementation will have some portions that are similar to the documentation, because both the implementation and the documentation must also include the exact method declaration, including all the elements of the declaration, such as the arguments and return values. However, a given API may have more than one implementation, *i.e.*, the underlying program logic for the implementation of the API will differ from implementation to implementation. But each of the implementations must necessarily share the elements of the method declaration — the package names and related method elements — in order to interoperate with each other. If these elements are not present and identical in different implementations of the same API, programmers will not be able to use the same names and structures when using the API, since it is these elements that allow each piece of the software to speak to each other. If these names or structures are changed, software that references these names will fail to function, because the software will not be able to find and access the functionality it needs. To see why using the same names and structures is important, it may be useful to analogize this to non-programming languages. It is only by having a common vocabulary of words like "truck" that people can speak to each other. If the language is changed, even slightly — as it is when a speaker of American English uses "truck" while a speaker of British

English uses "lorry" — then confusion arises. For programmers, similar confusion would occur if two different implementations of the same API used different names, arguments, and return values. For software, the result would be even worse than mere confusion, since computers cannot guess at what the original software meant to say. Instead, faced with a similar situation, software would fail to execute altogether. Different implementations of the same API must use these same elements, in order to avoid confusion, inefficiency, and incompatibility.

54. Other portions of the implementation not directly governed by the method declaration of the API (*i.e.*, what I have referred to as the method's program logic) will vary between different implementations, *i.e.*, the source code comprising the different implementations will be different. For example, different programming languages can be used to implement a particular API. In the case of Android, both the Java programming language and the C programming language were used to create code to implement the APIs at issue. Any two given implementations, other than the parts required for compatibility (*i.e.*, the elements of the method declaration), are not likely to be identical if they are written by different programmers or companies. However, because they are constrained by the API and practical considerations such as programming efficiency and the underlying hardware, some portions may appear similar. For example, since cars are constrained by the requirements to have a steering wheel, gas and brake pedals, and four tires, they will often be similar "under the hood" to the untrained eye, featuring an engine, drive train, and brakes. But an expert will be able to distinguish a V8 from a V6 or direct fuel injection from a carburetor. Similarly, the source code that is "under the hood" of an API implementation may appear similar to another implementation of the same API, in large part because of practical programming constraints. At the same time, implementations can occasionally look quite different if there are specific reasons for

such differences — the digital equivalent of choosing between a fuel-efficient but slow four-cylinder (or even a hybrid) versus a hungry but powerful V8.

55. Typically, the written form of the software is captured in a specification independent from any specific implementation. The specification is a written document that describes the API, including the method declaration (name, arguments, and return values) as well as specifying any requirements that the code must meet. Despite capturing important information about the API, it would be incorrect to say that the API is the specification, or vice-versa. The analogies to written words may again be useful — just as one does not say that the definition of a lion in a dictionary is, in fact, a lion, so the API's specification is not the API, but rather a description of the API which may then take different forms.

56. This abstraction and conceptualization of the API is what makes it possible for new implementations of APIs to be built. One of the key values of an API is that when improvements are made "behind the scenes," programmers who use the API do not need to know that the change has occurred; they should only notice that the program is now faster, more efficient, or more error-free. This can only happen because the programmers (and the software they built) used the high-level abstraction represented by the API (*e.g.*, the name) and did not work directly with the concrete, underlying implementation.

57. The full Android API documentation for the "abs" method (*available at* http://developer.android.com/reference/java/lang/Math.html#abs%28int%29) can help illustrate these issues:

```
public static int abs(int i)

Returns the absolute value of the argument.

If the argument is Integer.MIN_VALUE,
Integer.MIN_VALUE is returned.

Parameters

   i      the value whose absolute value has to be
computed.

Returns

   the argument if it is positive, otherwise the
negation of the argument.
```

The documentation's first line, "public static int **abs** (int i)," is the *declaration* of the method. The declaration is the formal statement of a method's structure, containing the method's name, the list of arguments it accepts, and the type of result it returns (the "return value"). I have already shown similar text in the short version of the documentation shown in paragraph 47. Here, both the input and the result are numbers ("int" is short for "integer."). There is effectively only one way to say this, and the only thing the programmer chooses is the name of the method ("abs") and the shorthand for the argument variable name ("i"). Presumably, Oracle named this method "abs" in part to increase efficiency and ease of learning for programmers who were familiar with other preexisting programming languages, since this name has been used for this function in many older programming languages, such as C.

The Android source code that implements the "abs" method documented above is:[2]

---

[2] Available at http://android.git.kernel.org/?p=platform/libcore.git;a=blob;f=luni/src/main/java/java/lang/Math.java;h=1da0b905c4af9aaf930adf5b8b80d92193b7c462;hb=HEAD#l100.

```
package java.lang;
...
/**
 * Class Math provides basic math constants and operations such as
trigonometric
 * functions, hyperbolic functions, exponential, logarithms, etc.
 */
public final class Math {
        ...
    /**
           * Returns the absolute value of the argument.
           * <p>
           * If the argument is {@code Integer.MIN_VALUE}, {@code
Integer.MIN_VALUE}
           * is returned.
           *
           * @param i
           *             the value whose absolute value has to be
computed.
           * @return the argument if it is positive, otherwise the
negation of the
           *          argument.
           */

    public static int abs(int i) {
        return i >= 0 ? i : -i;
    }
```

The first line, "package java.lang;" is the name of the package of API elements in which the abs method resides, and indicates that this file contains a class which is part of that package. "public final class Math" is also part of this organization, reflecting the class which contains the abs method. Both of these lines (which appear above in black), in this exact form, must be present in order to accurately implement the API, so all implementations of the Math.abs function will contain these two lines. (The variable name, here "i", is not part of the definition, and so can be different between different implementations without impacting compatibility.)

58. The lines of text that begin with asterisks (which appear above in blue) are programmer's *comments*. Comments do not provide functionality to the software or affect the compiled code that is distributed to users; instead, they document what the code does and explain it to other programmers. In this case, they describe to a programmer the function of this API, and may also contain information about how to use the API. These comments, in turn, are used to automatically generate the documentation for the method.

59. Finally, the actual source code for the method is shown here in green and red. It starts by repeating the declaration of the method — "public static int abs(int i)" (in green). It then presents the program logic for the method — the single line "return i >= 0 ? i : -i;" (in red). This red portion is what actually tells the computer how to perform the method's functionality. In this case, the program logic could be stated in English as "if the number we are given is greater than or equal to 0, return that number, and otherwise return that number but with the opposite sign." Because creating the absolute value is simple, this program logic is brief, but for more complicated methods many more lines of program logic may be needed.

60. Of this substantial amount of text that constitutes the implementation and documentation of the abs method, other than the required organizational lines I discuss in paragraph 57, only the single line "public static int abs(int i)" (the method's name and declaration, underlined above) is identical between this implementation of abs (in Android) and Oracle's implementation of abs (in the works at issue). This declaration identifies the method, matching the declaration in the documentation and specification. Use of the same declaration is necessary if the two implementations are to be compatible, and an essentially identical declaration is in fact present in any implementation of java.lang.Math.

61. Besides Oracle's open source implementation of Java (typically referred to as OpenJDK), the non-profit GNU Project has written a Java implementation called GNU Classpath, and the non-profit Apache Foundation has written a Java implementation called Apache Harmony. *See, e.g.*, GNU Classpath documents at http://www.gnu.org/software/classpath/docs/cp-hacking.html and Apache Harmony documents at http://harmony.apache.org/faq.html and http://harmony.apache.org/subcomponents/classlibrary/compat.html. Compatibility between these implementations is desirable for a number of reasons (discussed in more detail in paragraph 33) — primarily the benefit to software developers and consumers that results from having choice and competition between API implementation providers. For example, for compatibility and standardization reasons, the "abs" function discussed earlier has the following identical method declaration not only in Java and Android, but also in Harmony and GNU Classpath:

Java:	public static int abs(int a)
Harmony:	public static int abs(int i)
GNU Classpath:	public static int abs(int i)
Android:	public static int abs(int i)

The similarities are not limited to the abs method. Each of these projects implements the API packages at issue in this case, using the same package, class, and method names.

H. SUN AND ORACLE ALSO HAVE IMPLEMENTED AND DISTRIBUTED APIs FROM OTHER SOFTWARE

62. One way of seeing the distinction between API and implementation is by noting that companies such as Sun and Oracle have, in the past, implemented pre-existing APIs.

1. Sun Implemented and Distributed APIs from Previous Generations of Spreadsheets as Part of StarOffice and OpenOffice.org

63. Between 1999 and 2011, Oracle, and Sun Microsystems before it, developed and distributed the StarOffice and OpenOffice.org "Calc" spreadsheet software, and this software implemented and distributed APIs from previous generations of spreadsheets created by other companies. As I will explain in this section, the Calc spreadsheet software contains an API, and this API is in large part based on the APIs originally developed for older spreadsheet software, including Visicorp's Visicalc and Microsoft's Excel spreadsheet software. The implementation of the APIs in the Calc spreadsheet program allows spreadsheet models developed in Excel, for example, to also be useful and run in the StarOffice or OpenOffice programs.

64. Most spreadsheet programs provides "spreadsheet functions" that enable users to write small programs — called "macros" — that manipulate data in a spreadsheet cell. For example, the function "ABS" calculates the absolute value of the number in a given spreadsheet cell, the function "AVERAGE" calculates the average value of the numbers in multiple spreadsheet cells, and "NPV" returns the net present value of an investment. These functions or macros are used by people using spreadsheets to create models whether these people are professional software developers, engineers, lawyers, investment bankers, scientists, or hobbyists These functions and the macros that they are used by constitute an API, because they are a mechanism that allows creation of written programs that communicate with the spreadsheet software's functionality.

65. Spreadsheets created by different software companies frequently use function names and argument structures from older spreadsheet programs. For example, the first column of the following table shows the names of all the spreadsheet functions that were supported

in VisiCalc in 1979 — VisiCalc's API. The other columns of the table show the function names used to operate the same functionality in Lotus 1-2-3, Microsoft Excel, and OpenOffice Calc. (The additional API elements or functionalities added in the later programs are not shown in this table.) As the chart shows, the function names originally used by VisiCorp's VisiCalc in 1979 were then used by Microsoft Excel 2003 and Oracle's OpenOffice.org. This shows that the original VisiCalc API of 1979 is included to this day in the Microsoft and Oracle products, with only one exception (VisiCalc's "ERROR" function, which has been replaced by #N/A or #VALUE in Excel and Calc).

VisiCalc (1979)	**Microsoft Excel (2003)**	**Oracle OpenOffice.org Calc (Today)**
@ABS	ABS	ABS
@ACOS	ACOS	ACOS
@ASIN	ASIN	ASIN
@ATAN	ATAN	ATAN
@AVERAGE	AVERAGE	AVERAGE
@COS	COS	COS
@COUNT	COUNT	COUNT
@ERROR	#N/A or #VALUE!	#N/A or #VALUE!
@EXP	EXP	EXP
@INT	INT	INT
@LN	LN	LN
@LOG10	LOG10	LOG10
@LOOKUP	LOOKUP	LOOKUP
@MAX	MAX	MAX

VisiCalc (1979)	**Microsoft Excel (2003)**	**Oracle OpenOffice.org Calc (Today)**
@MIN	MIN	MIN
@NA	NA	NA
@NPV	NPV	NPV
@PI	PI	PI
@SIN	SIN	SIN
@SQRT	SQRT	SQRT
@SUM	SUM	SUM
@TAN	TAN	TAN

66. Attached as Exhibit C is a table showing the names of the spreadsheet functions in Microsoft Excel 2003 and Oracle's most recent version of OpenOffice.org Calc, which was prepared based on the publicly available documentation available for Microsoft Office Excel 2003 at http://office.microsoft.com/en-us/excel-help/excel-functions-by-category-HP005204211.aspx and for OpenOffice.org Calc at http://wiki.services.openoffice.org/wiki/Documentation/How_Tos/Calc:_Functions_listed_by_category. Five rows from Exhibit C are reproduced here for discussion purposes:

Microsoft Excel (2003)	**Oracle OpenOffice.org Calc (Today)**
AMORLINC	AMORLINC
AND	AND
	ARABIC
AREAS	AREAS
ASC	
ASIN	ASIN

67. Exhibit C shows that many of the functions that constitute the API of Microsoft Excel 2003 were also implemented in OpenOffice.org Calc. Functions on the left are implemented in Excel, and functions on the right are implemented in Calc. In this sample from Exhibit A, three of the functions (AMORLINC, AREAS, and ASIN) are implemented in both spreadsheets, while ASC is implemented only in Excel and ARABIC is implemented only in StarOffice.

68. As shown in Exhibit C, overall, of the 340 functions implemented in the Excel 2003 spreadsheet function API, 324 (95%) are also implemented in StarOffice.

2. Sun Implemented and Distributed APIs from Linux as Part of the Solaris Operating System

69. As I will explain in this section, since 1999, the Solaris operating system, developed and distributed by Oracle, and Sun Microsystems before it, has contained or been delivered with APIs that are based on the APIs originally developed by the developers of the Linux operating system. The implementations of these APIs in Solaris facilitates the use of programs developed in Linux environments to run on Solaris machines.

70. The BrandZ project, also known as Solaris Containers, was a software system that Sun implemented starting in 2004. BrandZ worked with other software, called a "brand," to translate a non-Solaris operating system's functionality into the Solaris functionality, so that software written for the other operating system would run on Solaris. Essentially, each brand helped "translate" communications that used the other operating system's APIs into communication with similar Solaris APIs. In particular, Sun developed a brand called the "LX Brand". The purpose of the LX Brand software was to "enable[] Linux binary applications to run unmodified on Solaris" ("BrandZ WebHome," *available at*

hub.opensolaris.org/bin/view/Community+Group+brandz/WebHome). In order to achieve this goal, several components of the Linux API are implemented by the LX Brand software, including signals, system calls, and the "/proc" interface. (*See* "BrandZ Overview" (*available at* http://hub.opensolaris.org/bin/download/Community+Group+brandz/WebHome/brandzoverview.pdf.)

71. For example, the "/proc" interface allows programs to interface with the Linux operating system by reading and writing the contents of files in a special directory called "/proc." Reading and writing these files allows a program to discover the status of the operating system and processes running on the operating system. A process can be a program or a part of a program that the user is running and it can be part of the operating system, e.g., it might facilitate communication over the Internet, with a printer, or allow one program to pass data to another program. In Linux environments and many Unix environments, every process has a number that identifies it, the so-called Process Identifier or PID. Processes also have names — for example the process that starts up the first process for the operating system has PID one, but the name 'init' and a process designated for cryptographic programs might have the name 'crypto' and would certainly have a different PID than the 'init' process. One representative element of the /proc interface, known as "/proc/[pid]/status," allows a program to communicate with the operating system about the status of a particular process. To initiate the communication, the program asks for the contents of the /proc/[pid]/status file — that is a file whose name is 'status' that is located in the directory/folder corresponding to the process identifier of a process, e.g., /proc/523/status is the file that gives the status of process 523. The operating system responds to a request for information about a particular process by

filling the file named 'status' with text that shows the status of the process. After such a request, the first eight lines of the /proc/[pid]/status file might, for example, look like this:

```
Name:   [the name of the process]
State:  [the status of the process]
Tgid:   [the “Thread Group ID” of the process]
Pid:    [the “Thread ID” of the process]
PPid:   [the “Thread ID of the process’s parent]
TracerPid:      [The “Thread ID” of the tracing process]
Uid:    [ID numbers of users involved in the process]
Gid:    [ID numbers of groups involved in the process]
```

72. The text on the lefthand side of the file (such as "Name: ") is part of Linux's API. These are always present in /proc/[pid]/status. The text on the right is the information about the specific process, and will be different each time /proc/[pid]/status is accessed. Changes to this layout (for example, changing "Name" to "ID" or "Reference") would break applications that use this API. As a result, if another operating system wanted to be compatible with this API, it would need to print "Name:," "State:," etc., in exactly the same manner as Linux prints it.

73. The following chart shows the values — taken directly from the respective publicly available source code — which Linux and Sun's LX Brand use to create the text on the left hand side of the /proc/[pid]/status file. In each entry in the chart, "\t" means "tab", "\n" means end of line, and the "%s" is replaced by the relevant information for the particular process, so that "Name:\t%s\n" becomes

```
Name:     [the name of the process]
```

when the /proc/[pid]/process file is accessed.

Linux	**LX Brand**
`"Name:\t%s\n"`	`"Name:\t%s\n"`
`"State:\t%s\n"`	`"State:\t%s\n"`
`"Tgid:\t%d\n"`	`"Tgid:\t%d\n"`
`"Pid:\t%d\n"`	`"Pid:\t%d\n"`
`"PPid:\t%d\n"`	`"PPid:\t%d\n"`
`"TracerPid:\t%d\n"`	`"TracerPid:\t%d\n"`
`"Uid:\t%d\t%d\t%d\t%d\n"`	`"Uid:\t%u\t%u\t%u\t%u\n"`
`"Gid:\t%d\t%d\t%d\t%d\n"`	`"Gid:\t%u\t%u\t%u\t%u\n"`
`"FDSize:\t%d\n"`	`"FDSize:\t%d\n"`
`"Groups:\t"`	`"Groups:\t"`

74. Each line of the chart is identical, and this demonstrates that the output of the /proc/[pid]/status API is the same between Linux and the LX Brand software, and therefore that (in this respect, at least) Linux and the LX Brand software are compatible. If these lines were different, then the resulting file would be different, and the LX Brand software would not be compatible with Linux. Other elements of the /proc interface are similarly implemented in the LX Brand software.

75. The LX Brand software also reimplements Linux kernel system calls. System calls are a part of an operating system's API; they allow programs written by users to access resources managed by the operating system, e.g., to read and write files to kill processes, or to allocate memory to use in a program. These resources are managed by the operating system, but programs written by users to run on the operating system need access to some of the resources to be able to run properly or at all. The LX Brand software provides an emulation function which translates the Linux system call to an

equivalent Solaris operating system call, for each of 317 Linux system calls. A relevant fragment of the publicly available source code that performs this translation, listing each Linux system call, and the LX Brand function that implements the system call, is attached as Exhibit D. This source code file indicates that there were 178 Linux system calls that were implemented as part of LX Brand (the other 139 system calls were either not supported or were able to directly use the equivalent Solaris system calls without translation). Each of the LX Brand implementations of the Linux system calls use the same name as the relevant Linux system call, with "lx_" prepended to distinguish them.

76. For example, the Linux system call "futex" was introduced to Linux beginning in 2002, and Solaris does not have a "futex" system call. In order to provide compatibility for Linux software running on the LX Brand, the LX Brand software provides an implementation of futex called lx_futex, which has essentially the same name as futex, takes similar arguments, and behaves similarly. The actual program logic that implements the LX Brand lx_futex function and the Linux system call are not similar, suggesting that they were independently created.

77. The 177 other system calls implemented by the LX Brand follow the same pattern: the Linux system call name, plus the lx_ prefix, is used to identify a function that takes similar arguments and behaves similarly to the Linux system call for which the function is named.

78. Sun's "LX Brand" software implements only a subset of the Linux operating system API, and so is not completely compatible with Linux. Sun's overview presentation states that it "support[s] a subset" of the /proc API and the "minimum needed" devices ("BrandZ Overview" at 23 and 24) and the design document notes that the "CLONE_PARENT" argument to the clone(2) system call is also only partially implemented (*see* "BrandZ

Design Doc", section 3.5.1 ("Linux Threading"), available at http://hub.opensolaris.org/bin/view/Community+Group+brandz/design). This partial implementation still aids compatibility and programmer efficiency, because it is still better for the programmer to use some of the APIs than to have to completely rewrite the software to use new APIs.

79. Solaris has also incorporated specific APIs from the Linux C Library ("glibc") into the Solaris C Library. For example, the "uucopy()" system call, according to Sun's BrandZ Design Doc, "seems to be generically useful, so the uucopy() will be implemented in [Solaris] libc" and, in fact, Solaris gained an implementation of the uucopy system call in 2006, shortly after BrandZ was introduced (*see* Solaris's common/syscall/uucopy.c).

3. ORACLE IMPLEMENTED AND DISTRIBUTED APIs FROM IBM AS PART OF THE ORACLE DATABASE SERVER

80. As I will explain in this section, the Oracle Database server distributed by Oracle since 1979 contains an implementation of the API originally developed by IBM for the "System R" database.

81. The System R database's SQL API was first described in an academic paper published by IBM employees in 1974 ("SEQUEL: A Structured English Query Language," DD Chamberlin, et al.), and elaborated in a subsequent paper published in 1976.

82. The 1974 SEQUEL paper defined the following API elements or functionalities:

```
SELECT FROM
WHERE
GROUP BY
```

```
SUM
COUNT
AVG
MAX
MIN
```

83. IBM supplemented the functions in the 1974 paper in a subsequent paper published in 1976 ("System R: relational approach to database management," M. M. Astrahan et al.), adding several new elements or functionalities to the SQL API: HAVING, ORDER BY, CURSOR, INSERT INTO, and DELETE.

84. Each of the API elements or functionalities referenced in paragraphs 82 and 83, and defined in the 1974 and 1976 papers, were implemented by Oracle in 1979 and are still present in current releases of the Oracle Database server. Because these API elements are implemented in the Oracle Database server, a command using the API elements "SELECT FROM … WHERE …" would also be able to operate, with minimal changes, with current Oracle Database servers, as it did with the original IBM System R software (*see* "Oracle SQL: The Essential Reference," David C. Kreines (2000), Chapter 1, "Elements of SQL," *available at* http://oreilly.com/catalog/orsqlter/chapter/ch01.html).

85. For example, because the Oracle system implements the API elements or functionalities defined in the 1974 paper, it will still execute commands written using the 1974 SEQUEL API. The 1974 paper gives this short command that uses elements defined in the 1974 paper:

```
SELECT NAME
FROM EMP
WHERE SAL
```

```
SELECT   SAL
FROM     EMP
WHERE    NAME = Bl.MGR;
```

86. Because this command was written using API elements (in bold) originally defined by IBM but later implemented in the Oracle Database server, this command should still function in a modern Oracle Database server, and indeed, some sources report that this exact command was used in early demonstrations of the Oracle database (*see* "Oracle SQL: The Essential Reference," David C. Kreines (2000), p. xiv and Chapter 1, "Elements of SQL").

I. BASIC EXAMPLE OF JAVA METHOD USAGE

87. When a programmer is writing an application, and wants to use a particular functionality, he must invoke the functionality by using the appropriate method. If a programmer writing in the Java programming language wants to use Java's square root functionality to find the square root of 25, he would do that by incorporating the following language in his program:

```
double result = Math.sqrt(25.0);
```

88. The *argument* 25.0 is passed to the method Math.sqrt when the method is called, and "5.0" is *returned* by the method. In this example, the return value is then stored in the variable named "result" for use elsewhere in the program.

89. To write this example, a programmer who had never previously used Java would likely have started by guessing that square roots were in the class "Math," looking at that class's documentation, finding the familiar "sqrt" method, and then reading the documentation

for that method to understand what result is returned and what special cases need to be considered in writing code. He would then write the fragment of code above, and in the future, having learned to use this part of the API, he would not likely have to refer to the documentation again.

90. Note that at no point does the programmer need to know how the program logic that is invoked by the sqrt method actually calculates the square root — it could use a Newton-Raphson method, logarithms, or another mathematical algorithm for calculating the square root. As previously noted in paragraph 34, these details are "encapsulated" — hidden behind the scenes. This focus on knowing and understanding the API name and functionality, rather than understanding how the method's underlying program logic works behind the scenes, allows programmers to work more efficiently.

J. THE APIS AT ISSUE ARE METHODS OF OPERATION

91. I understand that section 102(b) of the Copyright Act states, "In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work." I also understand that a method of operation has been described by the First Circuit as "the means by which a person operates something, whether it be a car, a food processor, or a computer." I further understand that the Ninth Circuit, citing section 102(b) of the Copyright Act, has stated that the functional requirements for compatibility are not protectable. Under either of those definitions, as I will explain in more detail below, it is my opinion that the Java API specifications at issue in this case are methods of operation.

92. As previously mentioned, in some computer languages, methods are referred to as "functions." Both the terms function and method suggest — correctly — that functions and methods are literally a functional way to operate software. Once the method that is part of the API is called and the right parameters are passed to it, the API invokes functionality provided by the underlying software system. This "operates" the underlying software system to create the return value, just as use of the car steering wheel makes the car (through the steering system) take action to steer the car, the "plus" button makes a calculator add two numbers, and the "print" command makes the operating system print a paper copy of a document.

93. That an API is a functional method of operation is implicit in the definition of an API: the entire purpose of an API is to allow one program to "interface" with another "application." This interfacing is not a social or creative chat, but a formal, functional command from one program to another: "Do this thing for me, and report back when you are done." The program in command is using the API to operate the underlying program; and the underlying program, likewise, is being operated by means of the API. In fact, it is typically difficult, if not impossible, to operate the underlying system in any way except through an API.

94. As demonstrated above in the example of Math.sqrt, using the name is necessary to invoke the underlying functionality. It is also the *only* way to invoke the underlying functionality — one cannot, for example, change "sqrt" to "square_root" in the example above and still expect the code to work. Nor could one change the number or type of arguments. Programs, unlike the human operators of our calculator and car analogies, are not flexible — they must be fed precise information in order to operate.

95. Like “Ctl+P”, the print icon, or the steering wheel, APIs provide mechanisms that operate underlying functionality, causing the libraries at issue to perform activities and return the information requested by the programmer — that is to say, by the “operator” of the software in question.

96. As an example of how APIs in Java are methods of operation, I will consider three classes in the java.util package that allow developers, and the software they write, to operate on dates and times: Date, Calendar and TimeZone. As the names convey, these classes are used by programmers to create, manipulate, and use calendars and dates in Java programs. As software is increasingly deployed throughout the world, it is important that developers be able to simply create and manipulate dates and times in a way that works across all cultures and time zones. These Java classes provide such an API. These classes are used together, and the methods in these classes mirror the functionality and operation non-programmers would expect if you needed to create and use dates and times. For example, to use dates and times, first a programmer has to record them. A Date allows that by representing a specific instant in time. While one might expect this would be something like January 22, 2009, a Date actually represents a specific millisecond on a specific day of a month of a year, and then provides methods that translate that millisecond into a particular date, automatically translating (if necessary) into other calendars (like the Chinese or Hebrew calendars). Having provided a way to store the date, a programmer manipulating dates would likely want to be able to perform a variety of actions on the date, and it is these actions that are most clearly methods of operation. For example, a programmer might want to know whether one Date comes “before” another, so that they could sort a list of files by time-of-modification, or display a list of songs arranged by date of recording. Not surprisingly, the Date class provides a method to test if one date comes before another called “before.” The code to

execute that test and determine if a Date A comes before a Date B is written as "A.before(B)." This method executes the test and returns "true" if A is chronologically before B, and returns "false" otherwise. The Date class also has methods to determine if a date comes after another date and if two dates are equal. The names of these methods, respectively, are after and equals, providing a clear example of how the form/name of the methods follow their function. The Date class also provides a getTime() method that returns the exact time in milliseconds. The Calendar class contains methods used to create a calendar, e.g., for a specific time zone, year, and/or a specific international location. The Calendar class has methods that allow the programmer to determine the first day of the week, which is SUNDAY in the United States, but MONDAY in France, for example. This method is Calender.getFirstDayOfWeek() and it returns a value such as Calendar.MONDAY or Calendar.SUNDAY — two values of the calendar class that are clearly functional in representing days of the week. These classes are typically used together with the TimeZone class, which provides convenient methods for creating and using the timezones that occur in the world. For example, the method TimeZone.inDayLightTime(d) determines whether the Date d is in daylight saving time in the given time zone.

97. Methods can also invoke more concrete functionality. For example, the class java.io.file gives programmers the ability to do operations on a file. This includes useful methods like "createNewFile" (which creates a file), "getName" (which gets the name of the file), and "delete" (which deletes the file). In each case, the programmer invokes the underlying functionality — such as creating or deleting a file — by using the name of the method. Once a file has been opened, the methods in the java.io class can be used to read and write the file; for example, use of the method name "readLine" will invoke software that reads a line from the file.

98. In each of these cases, whether operating on simple data like a date, or more complex things like a file or web page, the method names are the way in which the underlying functionality is invoked, providing information and taking actions as needed by the program. The method name is, quite literally, the method of operation, just as the gas pedal is the way that an engine is invoked.

K. THE JAVA API PACKAGE NAMES ARE DICTATED BY FUNCTION

99. It is my understanding that names are not entitled to copyright protection. However, even if they were, it is my opinion that the Java API package names are short, fragmentary, and functional. It is my understanding that short, fragmentary names that are dictated by function are not protectable under copyright law.

100. It is my understanding that the following API packages (and, in some cases, certain subpackages of these packages) are at issue in the case. In each case, the name of the package and the basic organization of the classes and methods within each package are merely descriptive of the functionalities in those packages.

java.lang

The java.lang package and its subpackages java.lang.ref, java.lang.reflect, and java.lang.annotation are part of a group of classes that facilitate interacting with and programming related to the Java language. The package name ("lang") and contents of the classes and methods in this package reflects this emphasis on the core Java language.

java.math

The java.math package provides the programmer with access to classes that facilitate arbitrary precision arithmetic with integers, *e.g.*, integers with no upper or lower limit.

The package name ("math") and contents of the classes in this package reflect this underlying functionality.

java.net

The java.net package, and its extension javax.net and subpackage javax.net.ssl, provide classes for the programmer to implement network connections at both a low- and high-level. The package name ("net") is short for "network" and therefore reflects this underlying functionality.

java.io and java.nio

The java.io and java.nio packages group together classes for dealing with input and output. Input and output are called "I/O" in long-standing programmer jargon, explaining the name of the io package. (The nio package is so-named because it was an attempt to present a "new" io (nio) package.) The java.nio hierarchy of classes also contains the subpackages java.nio.channels, java.nio.channels.spi, java.nio.charset java.nio.charset.spi. The nio package provides classes that facilitate more efficient (faster) input and output. The nio package are designed to interact with each particular operating system's efficient I/O mechanisms, so that the Java programmer can use the nio classes knowing that they will likely be faster than the java.io classes that were not originally designed for efficiency.

java.security and javax.security

The java.security package, its subpackages java.security.acl, java.security.cert, java.security.interfaces and java.security.spec, and its extensions, javax.security.auth, javax.security.auth.callback, javax.security.auth.login, javax.security.auth.x500, javax.security.cert, provide classes and functionality related to security, as the names

suggest.

java.sql

The java.sql package, and its extension, javax.sql, facilitates interacting with relational databases or data in a format similar to that defined in such a database. These packages are based on the "SQL" standard (Structured Query Language), a standard named and defined in the late 1970s and early 1980s, and the name of the packages ("sql") reflects the name of this standard.

java.text

The java.text package facilitates writing software to handle text, dates, numbers, and messages in a format that is independent of a particular natural language — allowing programmers to cope more easily with languages other than English.

java.util

The java.util packages provides utilities and collection classes. Its subpackages java.util.logging, java.util.jar, java.util.prefs, java.util.regex, and java.util.zip provide utilities that are more specialized, *e.g.*, to deal with logging, archives (jar files), user preferences, regular expressions, and zipped or compressed files, respectively. These functionalities are diverse, but "utilities" are a traditional name for small, single-purpose tools in the computing world, and so grouping these together under the name "util" is a straightforward mapping of functionality to traditional naming.

javax.crypto

The javax.crypto package and its subpackages javax.crypto.interfaces and javax.crypto.spec provide classes to write code that adheres to cryptographic protocols.

"Crypto" is common programmer shorthand for "cryptography" and so makes for a natural mapping of name to functionality.

javax.xml

The package javax.xml and its subpackages javax.xml.datatype, javax.xml.namespace, javax.xmlparsers, javax.xml.transform, javax.xml.transform.dom, javax.xml.transform.sax, javax.xml.transform.stream, javax.xml.validation, and javax.xml.xpath deal with XML—eXtensible Markup Language.

In some cases, a given package may require the functionality of another package in order to function correctly, much like the upper floors of a building need the lower floors of the building to remain standing. The final two packages listed below, while not themselves basic to the functionality of modern operating systems, must be present in order for the previously listed packages to operate correctly and provide their complete, intended functionality to users:

java.awt.font

The package java.awt.font allows programmers to interact with low-level font information.

java.beans

The java.beans package facilitates software interaction with JavaBeans — traditionally viewed as a reusable software component conforming to specific conventions so that the component can be manipulated with visual and graphical tools.

101. Because these names all describe specific functionalities, and (as will be discussed in paragraph 113) they are limited by design rules to short, fragmentary words and phrases, there is no meaningful creativity in the package names.

L. THE JAVA API CLASS AND METHOD NAMES ARE DICTATED BY FUNCTION

102. In the Android packages at issue, there are 472 public classes, 150 public abstract classes, and 176 public interfaces. A public class is a class that is accessible to programmers who are using the API; a private class is internal to the library and can only be used by other parts of the library.

103. API element names, such as class names, must be factually descriptive of the underlying functionality so that programmers can recognize, understand, and remember them when reading and writing a program. While this is formally enshrined in the Java Language Specification (see discussion in paragraph 113), the rule has more pragmatic roots that date back to the earliest computer languages. The core reason that API component names are short and reflect underlying functionality is that inventive and creative names only loosely tied to the functionality would be difficult for programmers to remember.

104. For example, the method "sqrt" is short, simple, and memorable for programmers — and possibly even for non-programmers; a reader of this report may not need to be reminded more than a few times that "sqrt" means "square root." It is technically possible to instead call the square root method "Steve," just as it would be possible to build a calculator whose buttons use colors instead of numbers and mathematical symbols. But such a calculator would be difficult to use; a user would have to memorize the colors and their mapping to the underlying numbers and symbols. That would take some time and effort — so much time and effort, in fact, that users are likely to use a traditional calculator instead. Similarly, when the name of a method does not reflect the underlying functionality (as in the case where a square root method is called "Steve") the method

would become difficult to learn and remember. As a result, in practice all API element names are simple and factually descriptive.

105. A method whose underlying functionality is to test to see if two file names are equivalent, for example, could be called "equals" or "equivalent" but likely not much else. Even the longest API element names, such as SQLNonTransientConnectionException, are still tied to the underlying functionality. In that case, the name has three parts which demonstrate the underlying functionality: the word "SQL" reflects that this relates to the SQL database language (a language that pre-dates Java, and was not created by Sun or Oracle), and the word "exception" reflects that this relates to an "exception" (similar to an error message). Both of these terms have been used in software programming for over 30 years, predating Java by some time. A programmer would recognize that the third part of the name ("Non-Transient Connection") reflects underlying SQL functionality that is a term of art from outside the Java language. Since the names of all of these underlying concepts are fixed, or nearly so, the name of the method reflecting these concepts is also necessarily inflexible. The fact that the name reflects the underlying functionality is not merely convenient — it is practically required to allow the system to be comprehensible to programmers.

106. Class names, like the method names discussed above, are highly functional, in many cases showing only small variations directly related to the class functionality. For example, consider the seven classes whose names end in Event as shown below. These seven classes come from three different packages.

Class Name	**From Package**	**What is the Functionality?**	**Why is it in this Package?**
HandshakeCompletedEvent	javax/net/ssl	A class describing an event that takes place once a Handshake is Completed.	"Handshakes" are part of the Secure Sockets Layer (SSL) networking protocol, and so this is grouped with other "net" and "ssl" features.
PreferenceChangeEvent	java/util/prefs	A class describing an event that takes place when a Preference Changes.	Utilities that user track Preferences must have a way to track what happens when the preferences change, and so this is grouped with "util" "prefs."
NodeChangeEvent	java/util/prefs	A class describing an event that takes place when a Preference Node Changes.	"Nodes" are a common way to organize data. Since these nodes are used to store user preference data, information about the nodes (including

Class Name	From Package	What is the Functionality?	Why is it in this Package?
			changes to them) are grouped with other "prefs"-related functionality.
SSLSessionBindingEvent	javax/net/ssl	A class describing an event that takes place when an SSL Session Binds.	As part of the implementation of the SSL networking protocol, this is grouped with other "net" and "ssl" classes.
ConnectionEvent	javax/sql	A class describing an event that takes place when a Connection occurs.	Because this is an SQL Connection, it is grouped with other SQL methods.
RowSetEvent	javax/sql	A class describing an event that takes place when an SQL RowSet is changed.	Because this is an SQL Rowset, it is grouped with other SQL methods.
StatementEvent	javax/sql	A class describing an event that takes place when an SQL	Because this is an SQL Statement, it is grouped with other SQL

Class Name	**From Package**	**What is the Functionality?**	**Why is it in this Package?**
		Statement is changed.	methods.

These names are functional in specifying the purpose of the methods. The noun part of each name that precedes Event describes the event, but there is essentially no creativity in choosing the noun. For example, the HandShakeCompleteEvent describes an event that takes place after the hand-shaking protocol in making an SSL connection has been completed. Similarly, the RowSetEvent class describes an event that takes place when a "rowset" is changed in an SQL database. The names have been chosen not because of deep introspection or creativity on the part of the author, but by simply describing what functionality is contained in the class.

107. Another example of class names that conform to simple rules describing the underlying functionality are the 18 classes whose names end with InputStream and 15 that end with OutputStream. These classes are part of a variety of packages — some are grouped with other input and output functions in java.io and java.nio, but several are part of the packages java.util, java.util.zip, java.util.jar, java.security and javax.crypto. The table below shows thirteen of the InputStream classes below and their corresponding packages. Again the names for the classes are functional and limited by the responsibilities of each class: the LineNumberInputStream class reads data while keeping track of line numbers while the CipherInputStream uses a cryptographic cipher for reading data.

Class Name	From Package	What is the Functionality?	Why is it in this Package?
FileInputStream	java/io	A stream of inputs from a file.	This is part of the basic input/output ("io") functionality.
PushbackInputStream	java/io	Adds the ability to push data back into an input stream.	This is part of the basic input/output ("io") functionality.
ZipInputStream	java/util/zip	A stream of inputs from a zipped file.	Reading from a zipped file is part of the basic zip file ("zip") functionality.
JarInputStream	java/util/jar	A stream of inputs from a "jar" file.	Reading from a jar file is part of the basic jar file ("jar") functionality.
LineNumberInputStrea m	java/io	Adds the ability to count the line number to an input stream.	This is part of the basic input/output ("io") functionality.
StringBufferInputStrea m	java/io	A stream of inputs from a string buffer.	This is part of the basic input/output

Class Name	From Package	What is the Functionality?	Why is it in this Package?
			(“io”) functionality.
CipherInputStream	javax/crypto	A stream of inputs that have been passed through a cipher.	Ciphers are part of cryptography, and so functionality to use ciphers is part of the cryptography (“crypto”) package.
InflaterInputStream	java/util/zip	A stream of inputs from an “inflater” that “inflates” a zipped file.	Inflaters are part of “zipping” a file and so this is part of util/zip.
FilterInputStream	java/io	Adds the ability to filter to an input stream.	This is part of the basic input/output (“io”) functionality.
ObjectInputStream	java/io	A stream of inputs from an object.	This is part of the basic input/output (“io”) functionality.
DigestInputStream	java/security	Creates a “messages digest” of a stream of inputs.	Message digests are part of certain security routines, so

Class Name	**From Package**	**What is the Functionality?**	**Why is it in this Package?**
			this class is part of the java/security packages.
ByteArrayInputStream	java/io	A stream of inputs from a byte array.	This is part of the basic input/output ("io") functionality.

108. Java class names are short, fragmentary words and phrases. This is true of other Java API elements, like method names, as well. Most Java API element names are short word phrases, typically of 1-3 words in length. (See paragraph 115 for more detailed statistics on method name length). One example is the class "PrintStream," which (not surprisingly) adds printing functionality to output "streams." Elements in the PrintStream class include the method named "append," which appends the argument to the output stream, the method named "print," which prints the stream, and the method named "close," which closes the stream. In fact, every method in the PrintStream class, with only four exceptions, is one word. Two of the exceptions are two words — "setError" and "checkError," which, as one would expect, set and check the error state of the output stream. The other exceptions are "printf" and "println" — abbreviations for "print formatted" and "print line." Besides being brief and fragmentary, these abbreviations have been in use by programming languages since the 1970s, in Algol and C.

109. Other classes, such as the SecurityManager class, have slightly longer names. In this class, three-word phrases (such as "checkPackageDefinition") are predominant and there

are some four-word phrases (such as “checkCreateClassLoader”). But even here, the naming follows a consistent pattern — 30 of the 40 methods are named check[SomeProperty], consistently describing their underlying functionality, which is to check the status of the property referred to by the method name. For example, “checkCreateClassLoader” checks to see if it is possible to create a new class loader.

110. Because many classes need the same functionality, and the names of the methods in question are dictated by functionality or by rules (see next section), it is not surprising that many of the names are repeated. The most common names in Oracle’s implementation of Java 1.5 are:

Method name	**Number of Times Repeated**	**Functionality?**
toString	194	Converts an object to a String.
equals	157	Tests to see if two objects are equal.
hashCode	147	Creates a “Hash Code” (a numeric representation) of a class.
run	139	Runs the code in the object.
read	96	Reads (typically to a stream of characters).
write	94	Writes (typically to a stream of characters).
remove	88	Removes something (exactly what is removed depends on the class).
get	74	Gets the value of an object.
close	72	Closes a stream.
size	68	Returns the number of items in a collection of items.
clear	61	Clears the content of the thing referenced.
clone	59	Clones the thing referenced.
TOTAL	**1249**	**These 10 method names are used by roughly 1/6 of the methods in Oracle’s**

		implementation of Java 1.5.

111. The organization of the methods into classes, like the organization of classes into package, is driven by functionality and the requirement that programmers be able to efficiently find and use these methods. This is why, for example, all of the math functions listed in the table in paragraph 39 are in the same java.lang.Math class.

112. Because these names all describe specific functionalities, limited by design rules to short, fragmentary words and phrases, there is no meaningful creativity in the class or method names.

M. THE JAVA NAMES ARE THE PRODUCT OF MECHANICAL RULES

113. Java API element names frequently repeat certain key terms and patterns, following mechanical rules laid out in the Java Language Specification and elaborated over time by practice. The rules provide suggestions for structure and naming, stating, for example, that "[m]ethod names should be verbs or verb phrases, in mixed case, with the first letter lowercase and the first letter of any subsequent words capitalized." Similarly, names of class types are to be "descriptive nouns or noun phrases." Java Language Specification, First Edition, Section 6.8 "Naming Conventions," *available at* java.sun.com/docs/books/jls/first_edition/html/6.doc.html#11186. Both of these rules are followed by all the examples shown in this report, except for those methods that are drawn from older programming languages (like "sqrt").

114. Additional word patterns crop up repeatedly throughout the Java APIs. "InputStream" and "ChangeEvent," cited above, are two examples affecting a few dozen names, but others go much further. For example, the Java Language Specification rules for method

names state that methods that return the value of a variable should start with "get," and method names that set the value of a variable should start with "set." Other rules require specific methods to be in many classes, such as "hashCode" and "toString." In Oracle's implementation of Java 1.5, nearly one-third of the method names at issue (2,578 of the 7,796 methods) are determined by these rules, including roughly 2,000 that begin with either "get" or "set," and 164 named simply "equals." Testing whether one thing is equal to another thing is an extremely common operation for programmers, and so it makes sense that many different methods for testing equality would exist. At the same time, it makes sense to make sure that the operation has the same name everywhere — it would unnecessarily complicate the learning process if it were "equals" in one place, "sameAs" in another place, and so forth. These constraints yield the resulting name ("equals") — which is wholly functional, dictated by efficiency constraints and not creativity.

115. An additional 2,347 method names were single words, like "run" or "add." The remaining 2,871 methods are not long or complicated — they are, on average, only 2.344 words "long" (counting a method name like locateURL as two words and findBestMatch as three words). In Android, of the 9297 total methods, 3220 are unique methods, 2676 or 28.8% are one word names, 2909 are required names (like the "get" and "set" examples above), leaving 3,712 other methods whose average word length is 2.41.

116. Following these mechanical rules and seeking to create consistency reduce the amount of creativity and work necessary to write the API, and, more importantly, reduces the amount of work necessary to learn and memorize the API. As a result, any good API design will have naming rules like the Java API naming rules contained in the Java

Language Specification, which result in names that are functional and primarily dictated by efficiency constraints.

117. It is also my opinion that the rules and naming conventions imposed by the Java Language Specification further constrains any creativity associated with the API names.

N. THE ORGANIZATION OF API ELEMENTS IS DICTATED BY FUNCTION AND DOES NOT REFLECT CREATIVITY

118. The same restrictions that apply to naming also apply to the organization of methods. Just as the name must be tied to functionality so that the API is easy to find and remember, the organization into related groupings must also reflect underlying functionality so that programmers can discover and use the elements efficiently. For example, methods related to security, such as AccessController.checkPermission and Signature.sign, are most sensibly organized into packages primarily related to security — java.security and javax.security.

119. The practical requirement that all API element names and package organizations be related to the underlying functionality restricts the packages in which any class can be placed, and restricts the classes in which any method can be placed. The fourth column in the tables in paragraphs 106, and 107 show how and why various classes fit into their respective packages, and various methods fit into their respective classes, which can help demonstrate this. For example, ZipInputStream is a class which allows creation of an “input stream” which can read from a compressed file (known as a .zip file). Because it relates to .zip files, developers will look for it with other classes and methods related to .zip files, and so it is grouped with them in an appropriately named subpackage called java.util.zip.

O. THE SAME ANALYSIS APPLIES TO THE NATIVE FILES

120. It is my understanding that Oracle has alleged that a number of files written in the C programming language are at issue. These files are part of the implementation of the Java API. For example, the file name java_lang_reflect_Array.c (which I understand is one of the files at issue) reflects the name of the java.lang.reflect.Array class. Similarly, the C function names within the file java_lang_reflect_Array.c reflect method names within the java.lang.reflect.Array class. As with the implementation files in the Java language discussed above, these files written in the C language must use these names and organizing principles in order to implement the API in a compatible and interoperable manner. If Android could not use these names in these files, Android could not create a compatible, efficient implementation of these APIs.

121. For example, the file java_lang_Class.c has these 29 C language functions:

Dalvik_java_lang_Class_desiredAssertionStatus
Dalvik_java_lang_Class_classForName
Dalvik_java_lang_Class_getClassLoader
Dalvik_java_lang_Class_getComponentType
Dalvik_java_lang_Class_getDeclaredClasses
Dalvik_java_lang_Class_getDeclaredConstructors
Dalvik_java_lang_Class_getDeclaredFields
Dalvik_java_lang_Class_getDeclaredMethods
Dalvik_java_lang_Class_getInterfaces
Dalvik_java_lang_Class_getModifiers
Dalvik_java_lang_Class_getNameNative
Dalvik_java_lang_Class_getSuperclass
Dalvik_java_lang_Class_isAssignableFrom
Dalvik_java_lang_Class_isInstance

Dalvik_java_lang_Class_isInterface

Dalvik_java_lang_Class_isPrimitive

Dalvik_java_lang_Class_newInstance

Dalvik_java_lang_Class_getSignatureAnnotation

Dalvik_java_lang_Class_getDeclaringClass

Dalvik_java_lang_Class_getEnclosingClass

Dalvik_java_lang_Class_getEnclosingConstructor

Dalvik_java_lang_Class_getEnclosingMethod

Dalvik_java_lang_Class_getGenericInterfaces

Dalvik_java_lang_Class_getGenericSuperclass

Dalvik_java_lang_Class_getTypeParameters

Dalvik_java_lang_Class_isAnonymousClass

Dalvik_java_lang_Class_getDeclaredAnnotations

Dalvik_java_lang_Class_getInnerClassName

Dalvik_java_lang_Class_setAccessibleNoCheck

Each of these functions corresponds exactly to a so-called native method in the class java.lang.Class. For example the C language function Dalvik_java_lang_Class_isAnonymousClass corresponds to the method AnonymousClass in the class java.lang.Class; the C function Dalvik_java_lang_Class_setAccessibleNoCheck corresponds to the method setAccessibleNoCheck in the class java.lang.Class, and so on for each function in the file.

122. The other .c files share similar characteristics to their corresponding .java files. The functions in the .c files typically correspond exactly to a corresponding public method in the related .java file or to a private method used to implement the private method. For example, in the file java_lang_Runtime.c ,the function Dalvik_java_lang_Runtime_nativeLoad corresponds to the private method nativeLoad in java.lang.Runtime which is used in the implementation of the public method

java.lang.Runtime.load. In my opinion, there is no expressiveness in the names used in the C files, because they are directly derived from the functional names in the .java files (as explained above), and are required for efficient implementation of those files.

P. MANY API ELEMENTS ARE DRAWN FROM THE PUBLIC DOMAIN AND ARE NOT ORIGINAL TO JAVA

123. It is my understanding that names are not entitled to copyright protection. However, even if they were, it is my opinion that many of the Java names are drawn from the public domain.

124. Java, like many other programming languages, is based on features of previous well-known languages, such as C and C++, including their grammar and syntax. *See, e.g.*, http://java.sun.com/docs/books/jls/first_edition/html/1.doc.html ("the lexical structure of Java . . . is based on C and C++"); *see also* http://www.gotw.ca/publications/c_family_interview.htm (James Gosling, inventor of Java, quoted as saying "You can go through everything in Java and say 'this came from there, and this came from there'"). Reuse of grammar and syntax from already-familiar languages allowed developers to leverage their existing knowledge and more quickly adopt Java. Similarly, authors of new programming languages often use old method and class names when appropriate, in order to help developers reuse their skills and transition to new languages, and to help make sure the ideas are time-tested. *See, e.g.,* James Gosling's "Feeling of Java" paper, *Computer*, Vol. 30, Issue 6, June 1997, where he writes "Java feels very familiar to many different programmers because Sun had a very strong tendency to prefer things that had been used a lot over things that just sounded like a good idea." As a result, many API element names in modern languages are drawn from the public domain. For example, package names like java.io, java.util, and java.net

reflect industry shorthand for common functionality like input/output, utilities, and networking, respectively. These packages with common names then, in turn, contain methods whose naming reflects industry custom and representation of the underlying functionality of the method. Some examples of functions that are very similar in Java and the pre-existing C and C++ languages as a result of their functionality and industry custom are shown below:

Name	**Originated in?**	**Java equivalent**	**What is it?**
char	At least C; *see* C Reference Manual, Dennis Ritchie, 1975 (*available at* http://www.cs.bell-labs.com/who/dmr/cman.pdf)	char	A data type holding a character; used repeatedly in method names, such as C++ "getchar" and Java "getChars." Other data types, such as *int* and *double*, also date back to C and at least the 1970s.
int abs (int i)	At least C; *see* C Reference Manual.	public static int abs (int a)	A function, returning the absolute value of the argument.
printf()	At least C; *see* C Reference Manual.	printf() (part of the java.io package)	A function, printing a formatted string to the screen or other output device.

125. Indeed, many of the names and concepts in Java have been used by the industry for decades. For example, the C Reference Manual references "int," "double," and "char," all used in Java. The "Bool" data type, which became "Boolean" in Java, dates back to at least Algol in 1968, and is a direct reference to Boolean logic — invented in the 1800s.

126. Another example of this is the java.util.regex API package, which implements "regular expressions" — a standardized way of testing if a given string of characters matches a particular pattern. Regular expressions were first formalized in 1968 ("Programming Techniques: Regular expression search algorithm," Ken Thompson, Communications of the ACM, Vol. 11, Issue 6, June 1968) and were known by the abbreviated name used by Java (regex) at least as early as 1983 (*see, e.g.* http://groups.google.com/group/net.lang.c/browse_thread/thread/6409987225e13a31/50da7fdd143184bd?q=regex#50da7fdd143184bd). Java.util.regex has two classes: Pattern, and Matcher. Method names in the Pattern class are compile, flags, matcher, matches, pattern, and split, while method names in the Matcher class include matches, pattern, reset, and start. Each of these names — particularly the extremely common "pattern" and "matcher" — are used in publicly available regular expressions software that predate java.util.regex, and all of them are discussed in *Mastering Regular Expressions*, Jeffrey E. F. Friedl, O'Reilly and Associates, 1997, which Oracle's documentation for java.util.regex cites. Sun also used third-party source code (Jakarta Regexp) to implement java.util.regex, and this code also had references to many of these terms, including "compile," "pattern," and "match."

127. Similarly, the java.sql and javax.sql packages are also based in part on pre-existing terms widely used in the industry. The package names themselves are a reference to the SQL standard, originally introduced in the academic literature as SEQUEL in 1974 (SEQUEL: A structured English query language, Proc. ACM SIGFIDET Workshop, May 1974, pp.

249-264). The classes and methods frequently are named after SQL concepts, and in particular (according to Sun's documentation at http://jcp.org/aboutJava/communityprocess/first/jsr054/jdbc-3_0-pfd-spec.pdf) on the X/Open SQL Call Level Interface (CLI), which dates to the first half of the 1990s (*available at* http://pubs.opengroup.org/onlinepubs/009654899/toc.pdf). For example, java.sql includes classes named "Array," "Blob," "Clob," and "Ref," which are the names of data types from the SQL standard. Similarly, the SQL CLI standard defines a method called "prepare" that operates on a StatementText. Java.sql's Connection class has a method called prepareStatement that has similar functionality. The SQL CLI standard also uses "commit" and "rollback" to discuss specific actions that can be done to a database, and java.sql's Connection class has matching "commit" and "rollback" methods that perform the actions discussed in the standard.

128. Java.util.zip is another example where the name of the package, and at least some API element names within the package, are references to terms that substantially predate Java's use of the terms. In this case, "zip" is a reference to the zip file format that has been in use since before the creation of Java. Class names in this package include "Adler32" (named after the Adler-32 algorithm invented by Mark Adler and licensed to the public as part of the zlib library) and "CRC32" (named after the CRC-32 algorithm, which dates back to the 1970s). Within the java.util.zip classes, method names include "deflate," "inflate," and "setDictionary," which are very similar both to general industry terms for these processes but also to the specific function names "deflate," "inflate," and "deflatesetdictionary" that are in the publicly available open source library (zlib) that predates, and is incorporated by, Java. (*See* http://www.zlib.net/manual.html)

Q. THE APIS AT ISSUE ARE NECESSARY FOR BASIC FUNCTIONALITY AND INTEROPERABILITY

129. In my opinion, the functionalities grouped into each of the API packages at issue (as listed above) are basic to most modern operating systems and particularly to mobile systems. As a result, it is necessary to include these functionalities in the Android platform. For example, the java.net package contains functionality relating to networking, and every modern mobile software platform must have networking functionality. If this functionality was not included in Android, Android would not be a competitive, modern platform.

130. Once Google decided to provide the ability for developers to write applications using the Java programming language, compatibility and interoperability with the existing body of software, tools, and knowledge about the Java APIs was an external factor constraining Android's options. This essentially required Google to include the APIs at issue.

131. Because Android is written primarily in the Java language (over which I understand Oracle does not claim copyright protection), Google was practically required to include the APIs at issue. There would be little benefit to merely using the same grammar and syntax; in order for existing code in a language to be compatible and interoperable with new software written in the same language, the API elements that constitute the language must also be present, and named and organized identically. Even the slightest changes to the names or organization of API elements will thwart compatibility and interoperability, because existing code that used those elements would not run properly, and programmers would have to learn new API element names. For example, I have previously discussed the method "sqrt," which computes the square root of a number. If the method were changed even the slightest (say, to "sqroot"), then existing source code written in the Java

programming language would not compile. Even more work on the part of the programmer would be required if, for some reason, the organization, arguments, or return values of the methods needed to be changed. It is not necessary to be a programmer to understand how this could be jarring; changing the shortcuts for copy and paste from Ctl+C and Ctl+V to something else would require every user of word processors to change their behaviors, which is why those two keyboard commands have been used unchanged across many programs since the first graphical user interfaces. As previously mentioned in paragraph 35, this sort of compatibility and interoperability is important to the industry, since (in the worst case) it allows programmers to reuse known, tested code fragments — an important practice in the industry — and in the best case, where compatibility is complete, it allows reuse of entire programs without modification. Therefore, the APIs at issue were included in Android in order to allow Android to be interoperable with existing code written in the Java programming language.

132. It is worth emphasizing that reuse of the API in this way does not mean that the underlying program logic implementing the API was copied.

133. In my opinion, the Java APIs are necessary for functionality, interoperability and programming efficiency.

R. THE APIS AT ISSUE ARE DEMANDED BY THE INDUSTRY

134. In my opinion, industry demand requires APIs that are compatible with Java, rather than APIs that are similar to, but not compatible with, Java.

135. Industry and developer practice would tend to make it very difficult for Google to choose a different API, or modify an existing Java API, when the Java language is used and supported by Google. While curious developers do teach themselves new APIs, as a

general rule they prefer not to be forced to retrain on new APIs unless there is an extremely good reason to do so. This is not simply a matter of losing the time spent learning; learning new APIs also means buying new books, losing the ability to reuse code fragments, and temporarily losing the fluency that comes with expertise in a particular language's idioms and structures. Indeed, a significant goal of the discipline of programming has always been to create reusable tools and build on what has been developed before, and the value of sensible reuse of existing APIs has been understood to be a significant part of this.

136. It is not just individual developers who strongly prefer existing languages. Companies also prefer to write programs in existing languages. Doing so allows them to reuse existing source code and tools; even where they cannot reuse entire programs, reuse of fragments of code is very common. Such code reuse helps make software better by allowing the reuse of tested, well-understood code, but it is only possible where platforms allow the same APIs to be used. For example, existing code written in the Java language which references the "abs" method would not run on a new platform unless the new platform supported this method and the class and package in which this method resides by implementing the API. As a result of these factors, the industry as a whole — both programmers and the companies who employ them — strongly prefer to work with APIs and API elements with which they are familiar.

137. It is not a coincidence that many software developers are familiar with Java APIs. Sun went to great lengths to encourage developers to learn and use Java. This began when Sun made the Java language, documentation, and API implementation available at no charge in 1996, apparently with the intent to ensure that programmers throughout the industry knew and had internalized the Java language, including these APIs. Sun also

worked extensively with educational institutions to help make the Java language a common tool for introductory programming classes. For example, there was active collaboration between Sun and the College Board in promoting Java as the language to be used in the Advanced Placement Computer Science (AP CS) exams developed by the College Board. These changes began just as the exam was switched from Pascal to C++ in 1999. Java was too large a language to be taught completely, so a subset of the classes and methods were identified as being an important part of teaching computer science. This led to the development of a group of classes that were part of the AP CS program and that mirrored the Java API exactly, e.g., instead of using java.lang, the AP CS program identified and used ap.java.lang with a corresponding documentation as part of the subset. At a similar time, Sun collaborated with the BlueJ group headed by Michael Kolling to develop an IDE (Integrated Development Environment) for novice programmers that was simple, but that used best practices that were part of Sun's official NetBeans IDE. The development of BlueJ led to a world wide adoption of BlueJ in many colleges and high schools that continues today with the use of BlueJ and its derivative GreenFoot which sees widespread adoption. In part as a result of that effort, I taught Java to my students for many years, and continue to do so. This effort created a large base of programmers who had learned the Java language and APIs, and would be reluctant to retrain on other languages because of their investment in Java. It is my understanding that Oracle, in its filings to the court, has claimed that the number of programmers who have invested time and effort to learn Java is in the range of 6-7 million. The investment of these programmers in learning the APIs is likely in the hundreds of millions of man hours.

138. In part as a result of Sun's substantial efforts to encourage programmers to learn and use the Java language, a large number of software applications have been written in the Java

language. The investment involved in creating these existing applications creates further industry demand for compatibility that would allow use of this existing software or code fragments on new platforms.

139. In conclusion, it is my opinion that the Java APIs have the following qualities: First, the APIs represent concepts and methods of operation. Second, the API names, method declarations, and organization are dictated by function or mechanical rules and do not reflect creative expression. Indeed, as noted, many API element names existed in the public domain and were not original to Java. Third, the APIs at issue are necessary for interoperability and efficiency reasons, and their use is driven by industry demand.

VI. THE ANDROID PLATFORM IS NOT VIRTUALLY IDENTICAL OR SUBSTANTIALLY SIMILAR TO THE JAVA PLATFORM

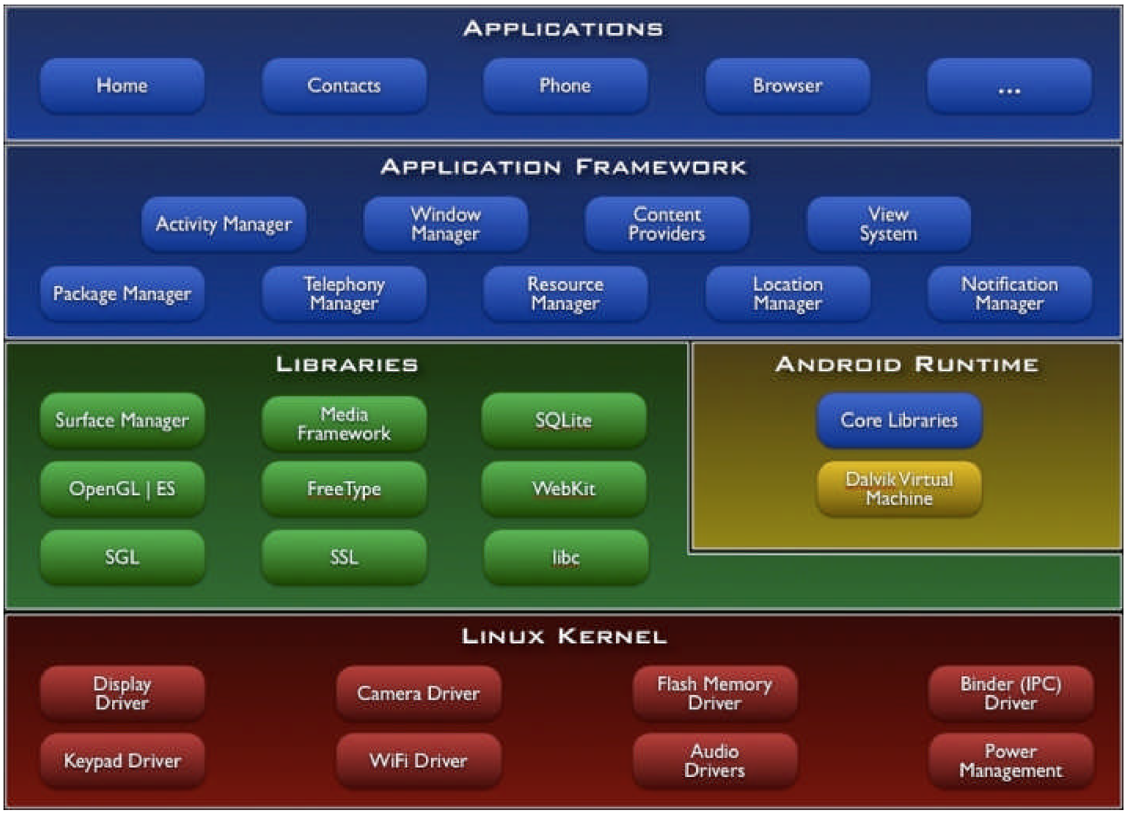
140. The Android platform uses the unprotectable API elements at issue as part of a larger overall framework that is substantially different from the Java platform, and more appropriate for the mobile platform.

141. The Java API packages which have been implemented by Android are a small part of the overall Android system, both in terms of the functionality they provide and the lines of code involved. The 48 APIs at issue are roughly one-third of the Android Runtime Core Libraries, which currently contains 168 API packages. The Core Libraries, in turn, are themselves only a small part of the overall Android architecture — they are the small blue box labeled "core libraries" in the yellow box at the right.[3] Other elements include the

---

[3] Diagram taken from the Android website, *available at* http://developer.android.com/images/system-architecture.jpg.

Dalvik Virtual Machine, the Linux kernel, a web browser, and a variety of other libraries and systems that are not part of the Java platform. These other components, when combined with the Android Core Libraries, make for a complete mobile operating system — something substantially different in scope and ability than the Java platform.



142. While the diagram above is not "to scale" (boxes of the same size may represent software of different size and complexity), an analysis of the number of lines of source code in the API packages at issue, in the Android Runtime Core Libraries, and in the other components in the diagram suggests that, if anything, the diagram likely overrepresents the size of the APIs at issue.

143. Using a Python script SlocCounter.py (attached as Exhibit E) based on the "sloccount" tool, a commonly-used tool for measuring the size of the source code of large software projects, Android's implementation of the APIs at issue in the "Gingerbread" release

constitutes 259,474 lines of code, in 1022 files. This is roughly 1.6% of the size of the entire Android source code, which comprises 57,076 files and 15,347,169 lines of code,[4] and roughly 15% of the 6,340 files and 1,713,087 lines of code[5] in the overall Android (Gingerbread) Runtime Core Libraries. Similarly, implementation of these APIs is a small portion of Oracle's JDK 1.5 implementation of the entire Java API, constituting 315,570 lines of code out of 2,867,712 (11% of the total) and 1001 files out of 9521 (10.5%).

144. Because Android's implementation of the APIs at issue comprises only 1.6% of the entire Android source code, Android as a whole is not virtually identical or substantially similar to the Java platform. The Java APIs at issue are not only a small portion of the Android platform, in my opinion the use of the Java APIs in the Android context is substantially different from the use of those APIs in the Java platform, which creates a different platform with different capabilities and functionalities targeting the emerging smartphone market rather than the desktop.

---

4 Numbers generated by running SlocCounterTotal.py (attached as Exhibit E) against a clean copy of Android (obtained following the instructions available here: http://source.android.com/source/downloading.html) and including only lines of code in .h, .c, .cpp, and .java files.

5 Numbers generated by running SlocCounterTotal.py against the libcore/ and frameworks/base/core/ directories in a clean copy of Android, counting only lines of code in .h, .c, .cpp, and .java files.

VII. ANDROID'S DOCUMENTATION OF THE APIS AT ISSUE IS NOT VIRTUALLY IDENTICAL OR SUBSTANTIALLY SIMILAR TO ORACLE'S DOCUMENTATION

145. Every API has documentation. This documentation is a written description of the functionalities provided by the API. The documentation is relied on by programmers when they need more detail and context than what is available from the name and organization of the method. Documentation also helps the programmer see what related functionality is available in the same classes and packages. Like the naming and organization of methods, good documentation of API components is extremely constrained, because it must be factual and succinct. Just as a square root function should not be named "Steve," the documentation of the square root function, in order to be maximally efficient for programmers who are referring to it, should be — and typically is — factual and strictly descriptive of the underlying functionality.

146. Because of the practical requirement that documentation for an API strictly describe the underlying functionality, and because programmers typically want to know the same important pieces of information about a given API, there are not many ways to write documentation for a specific API element. Two different authors documenting the same API components would necessarily write very similar documentation because they are describing the same functionality.

147. It may be useful to analogize writing documentation to writing a dictionary. If Merriam-Webster defines a zebra as an "African mammal [] . . . related to the horse but distinctively and conspicuously patterned in stripes of black or dark-brown" and Dictionary.com defines a zebra as an "African mammal[] . . . each . . . having a characteristic pattern of black or dark-brown stripes," this does not mean that the second

dictionary copied from the first. The definitions are similar because they are describing the same thing. It would be difficult to describe a zebra briefly without using words like "Africa," "stripes," "black," and "dark-brown." In addition, even where there may be multiple ways to describe things, the normal requirements that influence a dictionary (*e.g.*, clarity and brevity) are present for both dictionaries. This results in dictionary definitions that are similar even when written without copying.

148. Similarly, writing software documentation is significantly constrained, because the writer must convey the same factual information briefly, accurately, and with clarity. As a result of these constraints, skilled technical writers writing about the same API are likely to come up with descriptions that appear very similar and contain very similar descriptions of the critical features. The documentation for Java follows this pattern. For example, the method in the class java.io.PrintStream whose prototype is "void write(int OneByte)" takes one byte and writes (hence the name) that byte to the relevant "stream." Not surprisingly, Android's brief documentation of this method states that this method:

> Writes one byte to the target stream.

Oracle's brief documentation for the same method states:

> Write the specified byte to this stream.

The two different documentation writers have chosen different verb tenses for "write," and described the byte to be written slightly differently ("one byte" versus "the specified byte") but the descriptions are generally similar, not because they were copied from each other but because they must accurately and briefly describe the underlying functionality. Most documentation of the same method by two different authors will show the same pattern of similarity, because the authors are aiming at efficient, factual statements.

149. In my opinion, the Android and Java documentations are not virtually identical, nor are they substantially similar. Any similarity is dictated by the fact that they both document the same functionalities, and does not imply or demonstrate that they both involve the same creative expression. Indeed, the level of creativity reflected in the documentation is very low because it is intended to be highly descriptive, and is generally circumscribed by the nature of the functions described. It is my understanding that, in order to show that the Android documentation for the APIs at issue infringes Oracle's copyrights purportedly covering Oracle's documentation of the Java APIs, it must be shown that original and creative elements of the Java documentation, if any, have been copied into Android's documentation.

VIII. THE TWELVE FILES OR PORTIONS OF FILES ALLEGED BY ORACLE TO HAVE BEEN COPIED ARE QUALITATIVELY AND QUANTITATIVELY INSIGNIFICANT AND THEY ADD NO OR VERY LITTLE VALUE TO ANDROID

150. In this section, I will analyze the 12 files that Oracle alleges were copied by Google. These files represent, by number of files, 0.02% of Android and 0.13% of Oracle's implementation of Java 1.5 (12 files out of 57,076 in Android and 9,521 in Oracle's implementation of Java 1.5), and a similar percentage when measured by lines of code (742 lines out of 15 million and 2.8 million lines of code for Android and Oracle Java 1.5, respectively.) It is my understanding that, aside from these 12 files, Oracle does not allege that Google literally copied source code from Oracle.

A. TIMSORT FILES

151. It is my understanding that two Android files, TimSort.java and ComparableTimSort.java, are at issue. These files are a tiny fraction of a percent of Android and appear to have been donated by Google to Oracle, as explained below.

152. I have inspected three files: TimSort.java, ComparableTimSort.java, and Oracle's implementation of the Array class contained in Arrays.java. Only a single method out of the 11 methods in TimSort.java and 13 methods in ComparableTimSort.java is shared between those two files and Arrays.java. This method is called rangeCheck, and I have reproduced this method below:

Line #	**Code**
1	`private static void rangeCheck(int arrayLen, int fromIndex, int toIndex) {`
2	`    if (fromIndex > toIndex)`
3	`        throw new IllegalArgumentException("fromIndex(" + fromIndex +`
4	`                   ") > toIndex(" + toIndex+")");`
5	`    if (fromIndex < 0)`
6	`        throw new ArrayIndexOutOfBoundsException(fromIndex);`
7	`    if (toIndex > arrayLen)`
8	`        throw new ArrayIndexOutOfBoundsException(toIndex);`
9	`}`

Quantitatively, this method is 9 lines of code out of 3,179 lines in the Oracle JDK 1.5 version of Arrays.java, 9 lines out of 924 lines in the latest Android version of TimSort.java, and 9 lines out of the total 46,269 lines (0.5%) that compose Android's implementation of the *single* java.util package at issue.

153. Qualitatively, the rangeCheck method is also trivial. It merely performs a simple, utilitarian, and fairly mundane "sanity check," checking that certain arguments used

elsewhere are correct before they are used. The two arguments tested are the index to the first element of interest in an array ("fromIndex"), and the index to the last element of interest in the array ("toIndex"). They are compared against the number of elements in the given array ("arrayLen"), against zero, and against each other to ensure that their values are "in range" — that is to say that they are acceptable in the context of this code.

154. Because the first element should always come before the last element, if fromIndex is larger than toIndex — line #2 in the code, "if (fromIndex > toIndex)" — that means that the programmer made an error. If that happens, lines 3 and 4 of the method "throw an exception" (in programming jargon, they indicate that an error has occurred).

155. Similarly, because an index should, by convention, never be less than zero, if fromIndex is less than zero, this indicates another type of error. Line 5 tests for this ("if (fromIndex < 0)"), and if the test fails, line 6 signals the error by throwing another exception.

156. Finally, the last element of interest ("toIndex") must refer to an element that actually exists, so if toIndex is greater than the total number of elements in the array ("arrayLen") (line 7), that also indicates an error, and the another exception is thrown by line 8.

157. This code was also necessary for API compatibility with other sort implementations in Java. Because the "exceptions" thrown when an error occurs can be considered to be part of the API of a method, using code extremely similar to this was necessary for TimSort to be completely compatible with other sort implementations. If code extremely similar to this, throwing the same exceptions under the same circumstances, had not been used, the TimSort files could not have been accepted by Oracle into Java for use with Java 7.

158. It is my opinion that this single function is both qualitatively and quantitatively an extremely small portion of the functionality of the TimSort.java and

ComparableTimSort.java files, which are in turn an extremely small portion of the overall functionality of Java.

159. I have also reviewed the publicly available documents discussing the history of these two files. It appears that they were written by Google employee Joshua Bloch. While the files were first included in Android, it also appears that Google offered the files to Oracle (*see* http://markmail.org/thread/xwyxemce75vvz33h#query:+page:1+mid:vnipd7bqzs5vxfjw+state:results), and that this donation was accepted by Oracle (*see* http://blogs.sun.com/mr/entry/jdk7_m5). As a result of this donation, these files are now part of the most recent version of Oracle's implementation of Java.

B. SECURITY TEST FILES

160. It is my understanding that eight files in the directories "/support/src/test/java/org/apache/harmony/security/tests/support/acl/" and "/support/src/test/java/org/apache/harmony/security/tests/support/cert/" are at issue. These files are all what are known as "test files," and as explained below they add minimal if any material value to, and are a very small portion of, Android. Specifically, the files at issue are:

AclEntryImpl.java

AclImpl.java

GroupImpl.java

OwnerImpl.java

PermissionImpl.java

PrincipalImpl.java

AclEnumerator.java

PolicyNodeImpl.java

161. It is good engineering practice to write companion software that tests the functionality of the parts of the software that will be delivered to customers. This software (referred to as "test software") will do the software equivalent of factory testing, feeding the software test inputs (such as "2 + 2") and verifying that the correct output is returned (such as "4"). Individual tests are known as "unit tests," because they test a specific "unit" of the software, and the files that are used to implement these unit tests are often referred to as test files. Doing this testing in software, rather than manually, allows for the testing to be done quickly and reliably. The testing can be done, for example, after every single change to the software, rather than only every day or every week as might be required with manual testing.

162. These particular files are in directories labeled "test," and the structure of their source code indicates that they are part of a test package, instead of a package which is part of the publicly-available Android API. In addition, several of the files contain comments indicating that they are for "verification" of a specific interface. Verification of an interface is a common part of software testing. Finally, these files are referenced only in trunk/dalvik/libcore/security/src/test/java/tests/security/acl/IAclTest.java and other files in the same directory. This file, and others like it, appears to be the so-called "test framework" — the software that runs the tests. This code is clearly structured to call these files as tests, and not to use the files as part of the actual functionality of the Android platform. In addition, it is worth noting that Android's history shows that these

were removed from Android in January of 2011 and have not, as of this writing, been replaced.[6] This confirms that the files were immaterial.

163. These files are also in large part "dummy" files — instead of having complex logic, they return certain, fixed values, which is a common practice in test files. For example, one method in AclEntryImpl.java consists entirely of the following code:

```
public boolean isNegative() {
     return negative;
}
```

164. In a real (not test) file, the "isNegative" method would do some complex logic to understand whether the quality was negative. Here, because this is a "dummy" file used for test purposes, no logic or work is done — instead, it simply immediately returns "negative." This "dummy" result would not do much good for real code, but in a test environment, this predictability is useful — if a test shows that for some reason this function is returning "positive," then something is wrong and must be fixed. Dummy files in general, and these files in particular, are not particularly creative — given the functional constraint of the method names that they are testing, there is typically only one efficient, reasonable way to write them.

165. In general, test files are written for internal use by developers prior to a product's release. They are typically not distributed as part of consumer products, for two reasons. First,

---

[6] *See* changes at

http://android.git.kernel.org/?p=platform/libcore.git;a=commitdiff;h=6241c067e065065098eb50a7aef35a58f78447a6 and

http://android.git.kernel.org/?p=platform/libcore.git;a=commitdiff;h=95d52b3b1446af2fefd46f57efc1afb6c679e8cc

the testing is finished when the software is finalized since end users of consumer products cannot fix any problems found by the testing. Test files are simply not material to the customer experience. Second, distributing them takes up additional space and resources that could be used for other purposes that actually provide direct functionality to consumers, so test files can negatively impact the customer experience if distributed. As a result of these factors, test files generally are not used by or distributed to consumers. In my opinion, these test files are likely to be the same as others — not distributed to consumers and not material to the consumer experience.

166. Quantitatively, these eight files represent an extremely small part of Android's and Oracle's implementations of the Java APIs at issue. These codebases are roughly 1.7 million lines of code and 2.8 million lines of code, respectively, and so these eight files represent less than 0.1% of the lines of code. They are not even a substantial portion of the overall number of test files. My analysis of Android 2.3 suggests that there are at least 142 test files comprising 48,376 lines of code. These eight accused test files are less than 5% of this, measured in terms of lines of code.

167. Qualitatively, these eight files have no material effect on how Android implements the Java APIs at issue, because they are not used by or distributed to consumers.

168. In my opinion, these test files are qualitatively and quantitatively insignificant to the overall Android system.

C. COMMENTS IN CODESOURCETEST.JAVA AND COLLECTIONCERTSTOREPARAMETERSTEST.JAVA

169. It is my understanding that the files CodeSourceTest.java and CollectionCertStoreParametersTest.java are at issue. The comments in these files that are

at issue were not written by Google, add no value to Android, and are a very small portion of Android.

170. I have inspected these four files (two each from Oracle's implementation and Android's implementation), and the only things that appear to be the same between these classes are certain comments. Of the 36 comments in Android's CodeSourceTest.java at the time of the complaint, eight appear to be the same as comments in Oracle's implementation of the CodeSource class. Of the 16 comments in Android's CollectionCertStoreParametersTest.java, 12 appear to be the same as comments in Oracle's implementation of the CollectionCertStore Parameters class. No source code appears to have been copied, and the comments at issue appear to have been removed from Android.[7]

171. Comments are used in software source code because they help programmers understand the code that they are reading and potentially modifying. However, comments do not become part of the final product that is shipped to the user of the software. If all comments were removed, the functionality would be identical, and users would be generally unaffected. These comments, like all comments generally, would not have been distributed as part of any Android-based products; they would have only been available to any programmers who downloaded the source code from the Android website. It is my opinion, therefore, that these comments did not add material value to the Android platform.

---

[7] The removal is documented here:

http://android.git.kernel.org/?p=platform/libcore.git;a=commitdiff;h=a49d9caee4cd74c0d2cf83d79b8ecdc00453dff8

172. In addition, these comments are largely very descriptive and functional. For example, one of the copied comments appears to be from the following source code:

```
/**
  * Returns a formatted string describing the parameters.
  *
  * @return a formatted string describing the parameters
  */
public String toString() {
    StringBuffer sb = new StringBuffer();
    sb.append("CollectionCertStoreParameters: [\n");
    sb.append("  collection: " + coll + "\n");
    sb.append("]");
    return sb.toString();
}
```

173. The similar portion of the comment is "Returns a formatted string describing the parameters." This is a simple, declarative statement, which describes the source code below it. Like other documentation discussed in paragraph 145, there are very few ways to state this, because it is a simple, factual description of the operation of the public method below it. The other comments at issue, with one exception, are very similar — a single sentence factually describing the method in question. The one exception, slightly longer and more detailed, is still only three sentences long.

174. I have investigated the history of these files, and have determined that these comments were not created by Google. It appears that they were written by Intel employees who donated the files containing the comments to an open source project called Apache Harmony. My understanding is that Apache Harmony is an independent implementation of the Java APIs, created by a non-profit foundation called the Apache Foundation. My understanding is that Apache Harmony is licensed to the public under the terms of the Apache License, which allow anyone to reuse the code with essentially no restrictions. It is my understanding that, in compliance with the terms of the Apache License, Android reused the Apache Harmony implementation of some parts of the Java APIs, and my inspection of files in Android confirms this understanding. Android's versions of these

two files are virtually identical to Apache Harmony's versions of the same files, strongly suggesting that these two files (CollectionCertStoreParametersTest.java and CodeSourceTest.java) were obtained by Google for Android under license from the Apache Foundation.

175. The Apache Foundation makes its software records available to the public, documenting the history of all files distributed by the Apache Foundation, including CollectionCertStoreParametersTest.java and CodeSourceTest.java. By using these records, it is possible to see when a particular file was created, when individual lines of code were written or modified, and by whom. Using these records, I determined that these comments have been present in these files since before the Android project. For example, the Android file "CollectionCertStoreParametersTest.java" contained the comment fragment "the default parameter values (an empty and immutable." This same comment fragment is also present in the Harmony file "CollectionCertStoreParametersTest.java." Finally, this same fragment is present in the Oracle file "CollectionCertStoreParameter.java." By using the history features of the Apache Foundation's source code storage tool, I verified that this same fragment and matching surrounding text were present in that Harmony file when the file was initially created by an Intel employee, Geir Magnusson, on Jan. 8, 2006.[8] After the file was added to Android, Google did not change the comment; the comment stayed the same from the

[8] *See* file history available at http://svn.apache.org/viewvc/incubator/harmony/enhanced/classlib/trunk/java-src/security2/test/common/unit/java/security/cert/CollectionCertStoreParametersTest.java?limit_changes=0&view=markup&pathrev=367016.

time that the file was licensed from Harmony until they were removed.[9] The same is true of the other comments in these two files, all of which appear to have originated with Intel's contribution to Apache on Jan. 8, 2006. As a result, while I believe that these files had some identical comments (before they were removed by Google), it appears that this is related to choices made by Intel and not by Google.

176. It is also important to note that these two files represent an extremely small part of Android's and Oracle's implementations of the Java APIs. These codebases are roughly 1.7 million lines of code and 2.8 million lines of code, respectively, and so these two files represent less than 0.1% of lines of code.

177. In addition, these files are also test files, similar to the files discussed in the previous section. In my opinion, these files are qualitatively and quantitatively insignificant to the overall Android platform.

//

//

//

---

[9] *See* file history at http://android.git.kernel.org/?p=platform/libcore.git;a=history;f=luni/src/test/java/org/apache/harmony/security/tests/java/security/CodeSourceTest.java;hb=a49d9caee4cd74c0d2cf83d79b8ecdc00453dff8

178. I reserve the right to update and refine my opinions and analyses in light of any additional materials or information that may come to my attention in the future, including additional contentions by Oracle as well as any rulings issued by the Court in this case. I also reserve the right to supplement my opinions and analyses as set forth in this report in light of any expert reports submitted by Oracle and in light of any deposition or trial testimony of Oracle's experts.

DATED: July 29, 2011

Owen Astrachan, Ph.D.

EXHIBIT A: OWEN ASTRACHAN CV

Owen L. Astrachan

Department of Computer Science
Box 90129
Duke University
Durham, NC 27708-0129
telephone: (919) 660-6522
email: ola@cs.duke.edu

July 29, 2011

I. Education

Ph.D.	Computer Science	Duke University	1992
M.S.	Computer Science	Duke University	1989
M.A.T.	Mathematics	Duke University	1979
A.B.	Mathematics	Dartmouth College	1978
	with distinction in Mathematics, Summa Cum Laude, Phi Beta Kappa		

II. Professional Appointments

Duke University, Department of Computer Science

Professor of the Practice
July 2000 —.

Associate Professor of the Practice
July 1996 — July 2000.

Director of Undergraduate Studies
September 1993 — .

Assistant Professor of the Practice of Computer Science
1993 — 1996.

Lecturer in Computer Science
1991–1992. Developed and taught the introductory course for non-majors. Served on lab committee determining priorities for physical improvements.

Research Assistant
1991 (June–Aug). Research Assistant at SRI International, AI group, working for Mark Stickel on the design of intelligent and efficient automated reasoning systems.

Research Assistant
1988–1991. Research Assistant for Donald W. Loveland, investigating automated theorem proving. Designed and implemented an OR parallel theorem prover that runs on a BBN Butterfly GP-1000, TC2000 and on a network of Sun workstations.

Senior Graduate Instructor
1986–1988. Solely responsible for developing the curriculum and teaching the first course for majors in the Computer Science Department. Assisted with course in Operating Systems.

Teaching Assistant
1985-1986 Served as Teaching Assistant for the the first two courses for majors in Computer Science. Responsible for designing laboratory exercises and running recitation sections.

University of British Columbia, Computer Science Department

Visiting Scholar and Lecturer
Sept 1998 — June 1999 (on sabbatical from Duke)

Experience in Secondary Education

Math and Computer Science Teacher Durham Academy
1980–1985. Taught Advanced Placement Calculus, Advanced Placement Computer Science, Multivariable Calculus and Linear Algebra, Geometry, Introduction to Computer Programming, Finite Mathematics. Developed curriculum for Finite Math, AP Computer Science, and Multivariable Calculus.

Math Teacher Camp Lejeune HS, Camp Lejeune, NC
1978–1980. Taught honors Trigonometry, honors Geometry, Algebra I, Pre-Algebra.

III. Honors

2007, NSF, CISE Distinguished Education Fellow, *Interdisciplinary Problem- and Case-based Computer Science*, one of two inaugural CDEF awardees (see grants).

2004, IBM Faculty Award, *Issues in Large-scale Software Componentization*

2003, ACM International Collegiate Programming Contest (ICPC) Coaches Award.

2002, Richard K. Lublin Award for Distinguished and Motivating Teaching

2002, Nominated for Alumni Distinguished Teaching Award

2001, Nominated for Alumni Distinguished Teaching Award

1998, Outstanding instructor of Computer Science, University of British Columbia (teaching on sabbatical)

1997, NSF Career Award

1996, Nominated for Alumni Distinguished Teaching Award

1995, Duke University, Trinity College of Arts and Science: Robert B. Cox Distinguished Teaching in Science Award

1995, Sigma Xi

1994, Nominated for Alumni Distinguished Teaching Award

1978, A.B. degree awarded with distinction, summa cum laude, Phi Beta Kappa

IV. Publications

Journals:

Owen Astrachan and Robert Dewar. CS Education in the U.S.: Heading in the Wrong Direction. *Communications of the ACM.* July 2009, v. 52, n. 7, pp. 41-45.

O.L. Astrachan and D.W. Loveland. The Use of Lemmas in the Model Elimination Procedure. *Journal of Automated Reasoning*, v. 19 n.1, August, 1997, pp. 117-141.

Owen Astrachan, Kim Bruce, Robert Cupper, Peter Denning, Scot Drysdale, Tom Horton, Charles Kelemen, Cathy McGeoch, Yale Patt, Viera Proulx, Roy Rada, Richard Rasala, Eric Roberts, Steven Rudich, Lynn Stein, Allen Tucker, Charles van Loan. Strategic Directions in Computer Science Education. *ACM Computing Surveys.* v 28, n 4, December 1996.

O.L. Astrachan. METEOR: Exploring Model Elimination Theorem Proving. *Journal of Automated Reasoning.* v.13 n.2, 1994, pp. 283-296.

Books:

Owen Astrachan. *A Computer Science Tapestry: Exploring Programming and Computer Science with C++, Second edition.* McGraw-Hill, 2000.

Owen Astrachan. *A Computer Science Tapestry: Exploring Programming and Computer Science with C++*. McGraw-Hill, 1997.

Owen Astrachan. *The Large Integer Case Study in C++*. The College Board, Advanced Placement Program, 1997.

Book Chapters:

Owen Astrachan and Robert Duvall and Eugene Wallingford. Bringing Extreme Programming to the Classroom. in *Extreme Programming Perspectives*, Giancarlo Succi (ed.), Addison Wesley, 2002.

O.L. Astrachan and D.W. Loveland. METEORs: High Performance Theorem Provers Using Model Elimination. in *Automated Reasoning: Essays in Honor of Woody Bledsoe* ed. R.S. Boyer, Kluwer Academic Press 1991.

Miscellaneous:

O.L. Astrachan and Susan Horwitz and the Advanced Placement Computer Science Development Committee. *Communications of the ACM*. Technical Opinion: The First Course Conundrum. June, 1995. Pages 117-118.

Refereed Conferences:

Owen Astrachan. Pander to Ponder, *SIGCSE Technical Symposium on Computer Science Education*, Chattanooga, TN, 2009

Casey Alt, Owen Astrachan, Jeffrey Forbes, Richard Lucic, and Susan Rodger. Social Networks Generate Interest in Computer Science. *SIGCSE Technical Symposium on Computer Science Education*, Houston, TX, 2006.

Owen Astrachan. Non-Competitive Programming Contest Problems as the Basis for Just-in-time Teaching. *Proceedings of Frontiers in Education*, October 2004.

Owen Astrachan. Bubble Sort: An Archaeological Algorithmic Analysis. *SIGCSE Technical Symposium on Computer Science Education*, Reno, NV, 2003.

Owen Astrachan and David Bernstein and Andrew English and Benjamin Koh. Development Issues for a Networked, Object Oriented Gaming Architecture (NOOGA) Teaching Tool. *Proceedings of Frontiers in Education*, November 2002.

Owen Astrachan and Robert Duvall and Jeffrey Forbes and Susan Rodger. Active Learning in Small to Large Courses *Proceedings of Frontiers in Education*, November 2002.

Owen Astrachan and Robert Duvall and Eugene Wallingford. Bringing Extreme Programming to the Classroom, *Proceedings of XPUniverse*, Raleigh, NC, July, 2001.

Owen Astrachan. OO Overkill: When Simple is Better than Not, *SIGCSE Technical Symposium on Computer Science Education*. Charlotte, NC, February 2001.

Charles Keleman, Allen Tucker, Peter Henderson, Kim Bruce, Owen Astrachan. Has Our Curriculum Become Math-Phobic?, *SIGCSE Conference on Integrating Technology into Computer Science Education (ITiCSE)*, June 2000.

Owen Astrachan and Eugene Wallingford. Loop Patterns. *Pattern Languages of Programming (PLoP)*, Allerton Park, IL, August, 1998.

Owen Astrachan. Hooks and Props as Instructional Technology. *SIGCSE Conference on Integrating Technology into Computer Science Education (ITiCSE)*, August 1998.

Owen Astrachan, Geoffrey Berry, Landon Cox and Garrett Mitchener. Design Patterns: An Essential Component of CS Curricula. *SIGCSE Technical Symposium on Computer Science Education*. Atlanta, GA, February 1998.

Owen Astrachan and Susan Rodger. Animation, Visualization, and Interaction in CS 1 Assignments, *SIGCSE Technical Symposium on Computer Science Education*. Atlanta, GA, February 1998.

Owen Astrachan and Robert Smith and James Wilkes. Application-based Modules using Apprentice Learning for CS 2. *SIGCSE Technical Symposium on Computer Science Education*. San Jose, CA, February 1997, pp 233–237.

Owen Astrachan and Trevor Selby and Joshua Unger. An Object-Oriented, Apprenticeship Approach to Data Structures using Simulation. *Frontiers in Education*, Salt Lake City, Utah, 1996, pp 130–134.

Owen Astrachan and David Reed. AAA and CS1 : The Applied Apprenticeship Approach to CS 1. *SIGCSE Technical Symposium on Computer Science Education*. Nashville, TN, March 1995.

Owen Astrachan and Claire Bono. Using simulation in an objects-early approach to CS1 and CS2. *OOPSLA Conference Proceedings, Educator's Forum*. Portland, Oregon, October 1994.

O. L. Astrachan and D.W. Loveland. *METEOR*: Model Elimination Theorem Proving with Lemmas (system abstract). *Twelfth Conference on Automated Deduction* (CADE-12). Nancy, France, 1994.

Owen Astrachan. Self reference is a Thematic Essential. *SIGCSE Technical Symposium on Computer Science Education*. Phoenix, Arizona, March 1994.

Owen Astrachan. METEOR: Exploring Model Elimination Theorem Proving. *Workshop on Theorem Proving with Analytic Tableaux and Related Methods.* Marseilles, France, April 1993.

Owen L. Astrachan, Vivek Khera, and David Kotz. The Duke Internet Programming Contest: A Report and Philosophy. *SIGCSE Technical Symposium on Computer Science Education*. Indianapolis, IN, February 1993.

Owen L. Astrachan and Mark E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. *Eleventh Conference on Automated Deduction* (CADE-11). Saratoga Springs, NY, June 1992.

Owen Astrachan. Finding a Stable roommate, job or spouse: a case study crossing the boundaries of Computer Science Courses. *SIGCSE Technical Symposium on Computer Science Education*. Kansas City, MO, March 1992.

Owen Astrachan. Pictures as Invariants. *SIGCSE Technical Symposium on Computer Science Education*. San Antonio, TX, March 1991.

Owen Astrachan. METEOR: Model Elimination Theorem Prover for Efficient OR-Parallelism. *AAAI Spring Symposium on Representation and Compilation in High Performance Theorem Proving: Titles and Abstracts*, ed. W.W. Bledsoe, M. Stickel, P. Lincoln, R. Overbeek, and D. Plaisted, Stanford, CA, March 1989.

Unrefereed Reports:

Owen L. Astrachan and Donald W. Loveland *The Use of Lemmas in the Model Elimination Procedure*. Duke University Technical Report CS-1993-25.

Owen L. Astrachan. *METEOR: Exploring Model Elimination Theorem Proving*. Duke University Technical Report CS-1992-22.

Owen L. Astrachan. *Investigations in Model Elimination Based Theorem Proving*. Ph.D. Thesis. Also Duke University Technical Report CS-1992-21.

Owen L. Astrachan and Mark E. Stickel. *Caching and Lemmaizing in Model Elimination Theorem Provers*. SRI International Technical Note 513, December 1991.

V. Service

Professional Service

2009 – *College Board* HEAC: Higher Education Advisory Committee for Advanced Placement. Provide oversight and advice regarding the program.

2009, July, NSF Review Panel: Broadening Participation in Computing (BPC)

2009 *NITRD: Networking and Information Technology Research and Development Program*, panelist at public forum for discussion of the 2009 Federal Strategic Plan

2008 – *NSF/College Board* joint group on the First Year in Computer Science (chair).

2008, May, NSF Review Panel *CPATH*

2007 – *College Board AP Computer Science Redesign Commission* Committee charged with examining and redefining the Advanced Placement Computer Science Program.

2006– ACM Ed-Council. One of 25 members providing leadership and governance to the ACM about educational activities and outreach.

2005, July, NSF Review Panel *Advanced Learning Technology*

Program Committee OOPSLA 2005, Educator's Symposium

Program Committee OOPSLA 2004, Educator's Symposium

Internet & Society Idea Exchange Faculty Steering Committee for courses related to Internet and Society (oversight from Harvard and MIT, including 55 faculty from around the world).

NSF Review Panel CRCD Program
2004, reviewed proposals for the CRCD program in Computer Science at NSF.

ACM/College Board Digital Library Project 2004 – Advisory Committee to develop a project supporting Compute Science in high schools as part of the national NSF-sponsored digital library program.

ACM/College Board JETT Steering Committee 2002 – , Member of four-person steering committee providing oversight for joint ACM/College Board committee reviewing and approving national sites to host high school outreach programs for computer science.

Illinois Math and Science Academy
2002 – 2003, Member of three-person external review board for Mathematics/Computer Science at IMSA.

College Board/High School Computer Science AP Computer Science
2001 – 2002, Member of College Board *ad hoc* professional development committee to develop standards for training/educating high school teachers and workshop consultants in computer science.

NSF Review Panel CISE Program
2000, reviewed proposals for the CRCD program in Computer Science at NSF.

Math and Computer Science Mathematical Association of America
1999. Committee made recommendations to the MAA on the role of mathematics in computer science. Committee consisted of Alan Tucker, Charles Keleman, Dale Skrien, Charles van Loan, Peter Henderson, Kim Bruce, Owen Astrachan.

Chair, Advisory Committee for AP Computer Science College Board
1999–2000. Committee making recommendations on the use of new languages and curricula in Advanced Placement Computer Science. Committe consists of David Gries, Robert (Corky) Cartwright, Henry Walker, Ursula Wolz, Cay Horstmann, Fran Trees, Rich Kick.

External Oversight Board North Carolina Central University
1997 – 1998, oversee the growth and accreditation of the Computer Science Department.

NSF Review Panel CISE Program
1997, reviewed proposals to the Education Innovation program in the CISE directorate of NSF.

External Review Committee Oberlin College

1996, Member of external review committee to evaluate computer science department at Oberlin.

Program Committee CADE-13

1995/6, Member of program committee for CADE-13, Conference on Automated Deduction.

Chair, Advisory Committee for AP Computer Science College Board

1995–1996. Committee makes recommendations on the best use of C++ on the Advanced Placement Exam. The Committee is convened by the College Board, with representation from SIGCSE, the Computer Science Education special interest group of the ACM.

Judge, ACM International Programming Contest Association for Computing Machinery

1994–1997. One of six people responsible for developing problems and judging solutions for the ACM Programming Contest finals.

Chief Reader, Advanced Placement Computer Science Educational Testing Service

1989–1994. Responsible for developing grading standards and assigning scores for the AP exam in Computer Science. Assist with the development of the exam. Oversee, hire, and manage 70 University faculty consultants and High School educators in the grading of 10,000 AP exams taken by secondary students throughout the world.

Member, AP Computer Science Development Committee The College Board

1985–1989. Responsible for developing curriculum and devising tests for the AP exam in Computer Science.

Director, Duke Internet Programming Contest

1990 — 1994 . Co-founded a computer programming contest held in real-time over the Internet, involving 60 teams from 37 institutions in 5 countries (1990); 240 teams from 100 institutions in 9 countries (1991); 290 teams from 140 institutions in 14 countries (1992), 495 teams from 200 institutions in 20 countries (1993). Developed the problems used in the contest, designed solutions for the problems, and co-directed the administration of the contest.

IEEE Programming Contest

1994–1995. Coach of the Duke undergraduate IEEE Programming team. This competition is by invitation only to sixteen teams throughout the world. In 1994 Duke participated for the first time. In 1995 Duke won the contest.

ACM Programming Contest

1993–. Coach of Duke undergraduate ACM Programming Team. In 1994 the team won the mid-Atlantic regional contest and placed third in the world (first U.S. team) in the world finals. In 1995 the team won the mid-Atlantic regional contest and advanced to the world finals, finishing 22nd. In 1997 the team advanced to the world finals. In 1998 the team advanced to the world finals. (On sabbatical in 1999-2000.) In 2001, the team advanced to the world finals. In 2002 Duke had four teams in the top fifteen, and the top two in the Midatlantic regional contest; the top team advances to the world finals. In 2003 Duke advanced to the world finals. In 2004 Duke won the region (tied for first), advanced to the world finals and had three teams in the top 15 of the region; in the world finals Duke was one of four US teams that placed (above honorable mention) and was tied with Caltech and MIT for second among US teams. In 2005 Duke won the Region and advanced to the world finals receiving an honorable mention. In 2006 a Duke team advanced to the world finals, in 2007 Duke won the region and participated in the world finals, in 2008 Duke received an at-large bid to the world finals (to be held in 2009).

1989–1990. Member of Duke Programming team, 1989 and 1990 ACM International Programming Contests. Finished fourth in 1989 (Louisville, KY) and eighth in 1990 (Washington, DC) contest (world) finals.

Referee Activities

2010	SIGCSE	Technical Symposium on Computer Science Education
2009	SIGCSE	Technical Symposium on Computer Science Education
2008	SIGCSE	Technical Symposium on Computer Science Education
2007	SIGCSE	Technical Symposium on Computer Science Education
2006	SIGCSE	Technical Symposium on Computer Science Education
2005	ITiCSE	SIGCSE Conference on Integrating Technology into Computer Science Education (ITiC
2005	SIGCSE	Technical Symposium on Computer Science Education
2004	ITiCSE	SIGCSE Conference on Integrating Technology into Computer Science Education (ITiC
2004	SIGCSE	Technical Symposium on Computer Science Education
2003	SIGCSE	Technical Symposium on Computer Science Education
2001	SIGCSE	Technical Symposium on Computer Science Education
2000	SIGCSE	Technical Symposium on Computer Science Education
1996	SIGCSE	Technical Symposium on Computer Science Education
1995/6	CADE-13	Conference on Automated Deduction
1995	TABLEAUX '95	Workshop on Theorem Proving with Analytic Tableaux and Related Methods
	IJCAI	International Joint Symposium on Artifical Intelligence Automated Reasoning Track
	SIGCSE	Technical Symposium on Computer Science Education
1994	SIGCSE	Technical Symposium on Computer Science Education
1993	ILPS	International Logic Programming Symposium
	IJCAI	Automated Reasoning track
	ISMIS	International Symposium on Methodologies for Intelligent Systems
	SIGCSE	Technical Symposium on Computer Science Education
1992	CADE-11	Conference on Automated Deduction
	SIGCSE	Technical Symposium on Computer Science Education
1991	SIGCSE	Technical Symposium on Computer Science Education
1990	CADE-10	Tenth Conference on Automated Deduction

Duke Service

2009. Chair promotion committee for Jeffrey Forbes.

2008. Member of Office Education Committee (OEC) overseeing appointments of ROTC faculty.

2007-2009. Member of Academic Council.

2007-2010. University Committee on Admissions and Financial Aid (Academic Council).

2007–2009 Member of QEP (quality enhancement plan) University Committee to create a 75-page documentn outlining Duke's future as part of our 10-year SACS re-accreditation process.

2008 Search Committee, Dean to replace Robert Thompson, resulted in appointment of Lee Baker.

2007. Chair promotion committee for Susan Rodger.

2007-2009. University Commencement Committee.

2006-2007. Chair *ad hoc* Committee to Distingish Trinity College Degrees. Will report on the status of the BA and BS degrees at Duke.

2006–. Member of Faculty Research Committee that decides on Trinity College competitive grants and awards to faculty for research and conference activity.

2005. Member of Commitee on Departmental Support for Technology as part of the University committeess on Strategic Directions.

2005. Member of Arts and Science search commitee for the A&SIST Associate Dean.

2004-2005. Chair re-appointment committee for Prof. Richard Lucic

2004-. ISIS (Information Sciences and Information Studies) Faculty Steering Committee.

2004-. One of three faculty overseeing the development of teaching and learning, in conjunction with the CIT, as part of re-allocation of resources regarding the former center for teaching and learning.

2004-2007. Member Executive Committee of the Arts and Science Council (ECASC).

2004-2007. Member of ITAC, Committee on Information Technology at Duke.

Chair re-appointment committee for Prof. Richard Lucic 2002.

Member re-appointment committee for Prof. Jeffrey Forbes 2002.

Member Advisory Board for BlackBoard at Duke, 2002–

CITIE, IT skills committee, 2002.

Member Academic Integrity Council, 2001–

Member Executive Committee of the Arts and Science Council (ECASC) 2000–2003.

Interviewed candidates for A.B. Duke program, 2000, 2001

Member Board of Directors for Center for Instructional Technology (CIT), 1999–.

Arts and Science Council, 1999–2008.

Chair, Arts and Science Committee on Integrated Cluster Classrooms, 1999–2000.

Chair re-appointment/promotion committee for Prof. Robert Duvall, 2000.

Chair, University Planning Group on Instructional Technology, 1999–2000.

Search Committee, head of Career Development Center, 1998.

Member B.N. Duke Scholarship Committee, 1997

Member Core Team on evaluating use of Instructional and Information Technology, 1997.

ITAC Committee on Student Computing, 1996.

Chair search committee for Lecturer position in Computer Science, 1996.

Chair re-appointment/promotion committee for Prof. Susan Rodger, 1996.

Search Committee, Assistent Dean of Student Development, 1996.

Arts and Science Council, 1995–1996.

Member of Management Team (Center for Teaching and Learning) to develop an Exercise in Interactive Theatre for "Developing Teacher Knowledge", 1996.

Director of Undergraduate Studies, 1993–.

Provost's ad-hoc committee for Computer Technology and Education, 1995–1997.

Member of Steering Committee, Schulzbeger Interactive Learning Laboratory, Teaching and Learning Center, 1994–1996.

Faculty Advisor, DULUG: Duke University Linux User's Group, 1995–.

Departmental major advisor 1992– (supervise 20 first majors, 19 second majors per yaer.)

Premajor advisor at Duke University, 1986– 1998(ten first year students each year, total of 20 per year)

Chaired search committee for Assistant Professor of-the-practice, 1994.

Committee on the role of teaching for graduate students 1990–1992

VI. Consultancies

Expert Witness, Alston and Bird

2010 — Retained by Nokia in "Certain Mobile Communications and Computer Devices and Components Thereof," ITC Inv. No. 337-TA-704 (complaint filed Jan. 15, 2010), before the US International Trade Commission. Worked on Markman report, expert report, deposition and testimony.

Expert Witness, Alston and Bird

2008/2009 — Retained by Plaintiffs in Move, Inc., Nat'l Assoc. of Realtors, and Nat'l Assoc. of Home Builders vs. Real Estate Alliance Ltd et al., 2:07-CV-02185-GHK-AJW (filed Apr. 3, 2007) in the Central District of California. Worked on claim construction and expert report preparation.

Consultant, College Board
2008 – Oversee and help plan colloquium for college faculty (attended by 70 faculty) to understand current and future directions for AP Computer Science.

Google
2006 (six months) – Worked as an external contractor to help develop material used internally at Google for educating Google software engineers about Java and C++ programming.

Expert Witness, Womble, Carlyle, Sandridge and Rice
Software expert regarding work related to a contract dispute.

Consultant, AP Computer Science Educational Testing Service
1994 –. Advise development committee on incorporating C++ and Java into the Advanced Placement Program. Wrote Pascal/C++ case studies for use on the exam. Provided workshops for high school and college consultants in making transition to object oriented programming. Critique free response questions that are part of the national exam (2002, 2004).

Case-Study author, AP Computer Science Educational Testing Service
1994 –. Write the case study for use in the 1997-2000 AP exams. Write the code for the case study used in the 2001-2004 exams. A case is a "literate program", a treatise on the design, development, and implementation of a programming solution to a problem.

VII. Panels/Conference Acitivies

CS Principles: Piloting a New Course at National Scale, SIGCSE, Dallas, 2011 (with Larry Snyder, Tiffany Barnes, Dan Garcia, Jody Paul, Beth Simon).

The CS10K Project: Mobilizing the Community to Transform High School Computing SIGCSE, Dallas, 2011 (with Jan Cuny, Chris Stephenson, and Cameron Wilson)

The CS/10K Project, CRA/Snowbird Conference, Snowbird UTAH, July, 2010.

Code as a Metaphor for Computational Thinking, CSTA/CSIT Symposium, Google, Mountain View, CA, July, 2010.

Re-imagining the First Year of Computer Science, SIGCSE, Milwaukee, WI, 2010 (with Lien Diaz, Chris Stephenson, Jan Cuny, Amy Briggs)

FOSS Workshop, Free and Open Source Software, SIGCSE, Chattanooga, TN, 2009, invited speaker.

Computational Thinking Panel, SIGCSE, Chattanooga, TN, 2009 (with Amber Settle, Susanne Hambrusch, and Joan Peckham.

Advanced Placement Computer Science: The Future of Tracking the First Year of Instruction, Special Session, SIGCSE, Chattanooga, TN, 2009 (with Henry Walker, Chris Stephenson, Lien Diaz, and Jan Cuny)

Nifty Assignments, Special Session, SIGCSE, Chattanooga, TN, 2009 (with Nick Parlante)

Innovating our Self Image Special Session, SIGCSE, Portland, OR, 2008 (with Peter Denning)

Teaching Tips We Wish They Told Us Before We Started Special Session, SIGCSE, Cincinnati, OH, 2007 (with Dan Garcia, Nick Parlante, Stuart Reges).

Resolved: Objects Early Has Failed SIGCSE, St. Louis, 2005, Special Session (with Stuart Reges, Kim Bruce, Michael Kölling, Elliot Koffman).

But it Looks Rights: Bugs Students Don't See SIGCSE, Norfolk, 2004, Special Session (with David Ginat, Daniel Garcia, Mark Guzdial).

Colorful Illustrations of Algorithmic Design Techniques SIGCSE, Charlotte, 2001, Special Session (with David Ginat, Joseph Bergin, Dan Garcia).

Nifty Assignments in CS1 and CS2, Panelist, SIGCSE, Charlotte, 2001 (with Michael Clancy, Nick Parlante, Rich Pattis, Stuart Reges, Julie Zelenski).

FYI 2000: First Year Instruction, developed, organized, and chaired a workshop on first year instruction in computer science. The workshop had invited talks from both industry (Jon Bentley) and academia (Shriram Krishnamurthy, Richard Pattis) and was attended by more than 40 faculty from across the country. The workshop was sponsored by NSF, Microsoft, and the Duke Computer Science Department.

Patterns in Computer Science, (co-organizer with Eugene Wallingford), SIGCSE. Austin, TX. March 2000.

The Future of Advanced Placement Computer Science (panel). *SIGCSE Technical Symposium on Computer Science Education.* Austin, TX, 2000. With Corky Cartwright, David Gries, Cay Horstmann, Richard Kick, Fran Trees, Henry Walker, Ursula Wolz.

Nifty Assignments in CS1 and CS2, Panelist, SIGCSE, New Orleans, 1999 (with Michael Clancy, Nick Parlante, Rich Pattis, Stuart Reges, Julie Zelenski).

Incorporating Patterns into CS courses and *Writing Patterns for CS Courses* (co-organizer with Eugene Wallingford). SIGCSE, New Orleans, March 1999.

Future Directions in CS2 and Data Structures. Organized and ran the workshop that was held in conjunction with OOSPLA-98, Vancouver, CA, October 1998 (20 participants)

Object Oriented Design. Invited Participant, OOPSLA-97, Atlanta, Georgia, 1997.

Teaching Object-Oriented Programming: Practical Examples Using C++ and Java, Tutorial at PLDI 97, Las Vegas, Nevada, June 1997.

Future Directions in Data Structures and CS2. Organized and ran two-day workshop held at Duke co-sponsored by NSF, 32 participants, March, 2000.

Teaching Object-Oriented Design in the first year. Invited speaker and participant. OOPSLA-96, San Jose, CA, October, 1996.

Strategic Directions in Computing Research (SDCR), working group in Computer Science Education. Sponsored by ACM, CRA, and NSF, Boston, MA, June, 1996.

How to teach C++ in Introductory Courses, Tutorial part of PLDI, FCRC 1996, Philadelphia, PA, sponsored by SIGPLAN

Formal Methods Considered [Help — Harm]ful: Engaging students in the first year. Exploring Formal Methods in the Early Computer Science Curriculum, Joint NSF/US Department of Education Workshop. September 16, 1995 (invited speaker).

Developing an Object-Oriented Class Library. NSF sponsored workshop. Colgate University, June 1995. (invited participant)

A Sorcerer's Apprentice Approach to using C++ in CS1. NECUSE (New England Consortium on Undergraduate Science Education) Workshop in Introductory Computer Science Curricula. January 1995.

Measuring Performance of Automated Theorem Provers (with D. W. Loveland). *Twelfth Conference on Automated Deduction* (CADE-12), Workshop on Evaluation of Automated Theorem Proving Systems. Nancy, France, 1994.

Acquiring Object-Oriented Technology: A Bridge between Industry and Academia (invited participant). US West, Boulder Colorado, March 1994.

OOP: An introduction for Secondary School Teachers. Workshop delivered to secondary school computer science teachers in Dallas, TX, August 1993.

Simulation: A vehicle for exploring OOP. *Object-Oriented Curriculum Development Workshop.* NSF sponsored workshop. Colgate University, July 1993. (with Claire Bono)

A Tapestry of Fundamental Ideas and Concepts in Computer Science: a Programming, Contextural View for Liberal Arts Students *L**3: Logic, Loops, and Literacy.* NSF sponsored workshop on computer science for non-majors, Brooklyn College, May 1993.

Human Interaction with a High-Performance Theorem Prover. *International Joint Conference on Artificial Intelligence.* Workshop on Automated Theorem Proving. Chambery, France, 1993. (with D.W. Loveland)

Logic Programming Considered Harmful? *Joint Internatinal Conference on Logic Programming.* Prolog as a first language track, Washington DC, November 1993.

Caching to reduce redundancy in Model Elimination Theorem Provers. *Joint Japanese-American workshop in Automated Theorem Proving*, Argonne National Labs, June 1991.

The METEOR implementations of the Model Elimination procedure. *Workshop on Proof Theory and Automated Theorem Proving*. Oberwolfach, Germany, April 1991. (with D.W. Loveland)

Online Exams in Advanced Placement Computer Science. *National Council of Teachers of Mathematics Conference*. San Francisco, April 1984.

VIII. Invited Lectures, Talks, and Workshops.

Code as a Metaphor for Computational Thinking, Harambeenet workshop, Durham, NC, July 2010.

CS Principles and the CS10K project, NSF, BPC Community Meeting, Los Angeles, February, 2010.

CCSC: Midwest, Keynote Speaker, "A New Way of Thinking about Computational Thining", St. Xavier University, Chicago, October 2009.

National Academies: Workshop on Computational Thinking February 2009.

Problem-Centric Learning, Sept 2008, Rochester Institute of Technology

What is Computer Science?, April 2008, NSF CPATH, Living in the Knowledge Society.

Problems in AP Computer Science, June 2008, Advanced Placement Computer Science Reading, Professional Development Night.

CPATH: Science and Computer Science Purde University, November 2007.

CPATH: Problem-based Learning Keynote, Workshop sponsored by Argonne Labs and Governors University, November 2007.

Microsoft Computational Thinking Summit Redmond, WA, September 2007.

Google Faculty Summit, Mountain View, CA, July 2007.

HarambeNet: Introducing Computer Science through Modeling and Analysis of Social Networks SIGCSE 2007 Workshop, with Jeffrey Forbes.

The Cruelty of Really Teaching Computer Science Redux, University of California, Riverside, January 2006.

The Cruelty of Really Teaching Computer Science Redux, University of British Columbia Computer Science Distinguished Lecturer Series, Fall 2005.

The Cruelty of Really Teaching Computer Science Redux, University of Washington Computer Science Distinguished Lecturer Series, Fall 2005.

The Cruelty of Really Teaching Computer Science Redux, Keynote talk at CCSC/SE, Consortium of Computing Sciences in Colleges, Southeast US, November 2005.

The Cruelty of Really Teaching Computer Science Redux, Keynote talk at CCSC/E, Consortium of Computing Sciences in Colleges, Eastern US, October 2005.

A Random Walk Through Computer Science, Invited/Keynote Talk at ACM/Student conference Reflections/Projections, University Illinois, Oct. 2004.

20 Years of Teaching Computer Science, invited talk at NSF Workshop for high school teachers, Stonehill College, October 2004.

Everything I Needed to Know about Programming and Computer Science I Learned from my Teachers, Keynote Talk, SIGCSE, Norfolk 2004.

Using Patterns in the First Year, invited tutorial and presentation as part of the 2000 Eastern Small College Computing Conference, University of Scranton, PA.

OO Design and Patterns, invited speaker at NSF-sponsored workshop for high school teachers at Stonehill College, June 2000.

Advanced OO Programming, invited speaker at NSF-sponsored workshop for high school teachers at Stonehill College, January 1999.

Object-Oriented Design and Programming. Three-day lecture/workshop co-taught with David Gries delivered to educators from business colleges in Denmark, October, 1998.

Possible Futures for CS2 (panelist). *SIGCSE Technical Symposium on Computer Science Education*. Atlanta, GA, 1998.

Teaching C++ in AP Courses: Four day workshop designed and delivered for the College Board, June and August, 1997.

The First Computer Science Course and C++: Paradigm Lost or Regained. DIMACS workshop on Discrete Mathematics, July, 1996.

C++ in the Advanced Placement Program. AP Computer Science Reading, Professional Night, Clemson, SC. June, 1996.

Use of C++ for CS1 and CS2, Computing Science Conference, Philadelphia, PA, 1996.

The First Year: Beyond Language Issues, (moderator and proposer), *SIGCSE Technical Symposium on Computer Science Education*. Philadelphia, PA, 1996.

Advanced Placement and C++: Opening a Dialogue, (moderator and proposer), *SIGCSE Technical Symposium on Computer Science Education*. Philadelphia, PA, 1996.

Object-Oriented Programming: How to "Scale Up" CS1. *SIGCSE Technical Symposium on Computer Science Education*. Phoenix, Arizona, 1994.

Themes and Tapestries: A Diversity of Approaches to Computer Science for Liberal Arts Students. *SIGCSE Technical Symposium on Computer Science Education*. Phoenix, Arizona, 1994.

Using Case Studies in Computer Science Courses. *SIGCSE Technical Symposium on Computer Science Education*. Phoenix, Arizona, 1994.

On Computer Science and Teaching Computer Science with some perspectives from automated Reasoning. Bryn Mawr College, January 1993.

Faster, Fairer and More Consistent Grading Techniques: Lessons From the Advanced Placement Reading *SIGCSE Technical Symposium on Computer Science Education*. Washington D.C., 1990.

The Pleasures and Perils of Teaching Introductory Computer Science. *North Carolina Council of Teachers of Mathematics Conference*, November 1990.

Teaching Recursion in Introductory Computer Science Courses. *North Carolina Council of Teachers of Mathematics Conference*. November 1986.

IX. Professional Affiliations

Member of ACM, IEEE Computer Society, SIGPLAN, SIGCSE, SIGACT, SIGCHI, SIGART, SIGSOFT, Sigma Xi.

X. Research Funding

2009, NSF IIS, Special Projects: *Using Computational Thinking to Model a New Course: Advanced Placement Computer Science: Principles*, \$2,093,450.00, funding to College Board, PI.

2008, NSF BPC, Computational Thinking and Fluency in the 21^{st} Century. \$98,415. Submitted by College Board, PI.

2007, NSF CPATH, CISE Distinguished Education Fellow *Interdisciplinary Problem- and Case-based Computer Science*, \$250K over two years, one of two inaugural CDEF awardees in Computer Science.

2004 IBM Faculty Fellow, *Issues in Large-scale Software Componentization*, \$40K grant.

2002-2003 \$28K, IBM Echelon: Eclipse Help for Learning Online.

2002 \$400K. IBM SUR, Education/Research Grant for establishing COD (Cluster On Demand) (co-PI with Richard Lucic, Amin Vahdat, Jeff Chase).

2002 \$1,500, OOPSLA Educator's Grant: Using Design Patterns (declined)

2001 \$1,500, OOPSLA Educator's Grant: Using Design Patterns (declined)

2001 \$120,000 IBM Education/Research Grant for establishing a teaching/cluster classroom (co-PI with Susan Rodger, Richard Lucic, Kishor Trivedi, Amin Vahdat, Jeff Chase, the \$120,000 is just the education part of the grant, each of three groups was awarded a similar amount. This was part of a \$1.7 million SUR grant from IBM, some of which was software and related support.)

2000-2005, $480,826, NSF, *Modules and Courses for Ubiquitous and Mobile Computing*, NSF CRCD 0088078, PI, co-PIs Prof. Carla Ellis, Prof. Amin Vahdat.

2000, $1.19 million Microsoft *Interactive Research/Teaching Classroom*, with Jeffrey Vitter, Richard Lucic, Jeffrey Chase, Carla Ellis,Deitolf Ramm, Susan Rodger, Amin Vahdat. This includes $750,000 in software support and the rest in equipment, construction, and staffing support.

1998 $223,179 equipment *Establishing Interactive Collaborative Classrooms* Hewlett-Packard University Grants Program, co-PI with Susan Rodger

1998 $1,500, OOPSLA Educator's Grant: Using Design Patterns

1998 $50,000 Microsoft Eductional Development Grant, co-PI with Susan Rodger and Jeffrey Vitter

1998–2001 $150,306 U.S. Dept of Education GANN (co-PI with Jeff Chase, Carla Ellis, Alvin Lebeck, Jeffrey Vitter)

1997–1998 $80,000 equipment (part of a $1.6 million grant) from Intel supporting computer science education at Duke.

1997 $1,200, OOPSLA Educator's Grant: Using Design Patterns

1997–2002, $200,004, "Using and Developing Design Patterns", National Science Foundation: CAREER Program, CAREER #9702550

1996, $1,700, OOPSLA Educator's Grant: Using Design Patterns

1996–1997, $119,382, "The Applied Apprenticeship Approach (AAA): An Object-Oriented/Object-Based Framework for CS2", National Science Foundation Course and Curriculum Development, grant#DUE-9554910.

1996–2001, $607,800, "CURIOUS: The Center for Undergraduate Education and Research: Integration Through Performance and Visualization", NSF CISE Educational Infrastructure Program, grant #CISE-9634475 (co-PI with Prof. Susan Rodger)

1996–1998, $13,080 "U.S.-Germany Cooperative Research to Enhance the Performance of the Model Elimination Proof Procedure" (co-PI with Prof. Donald Loveland), National Science Foundation INT-9514375

1995, $1,000, OOPSLA Educator's Grant: Design Patterns in the Introductory CS Curriculum.

XI. Research Interests

Problem-centric Learning,Software Architecture, Object-oriented systems and languages, Computer Science Education, Networked and Distributed Computing, Automated Theorem Proving, Automated Reasoning, Parallel and Distributed Computing.

XII. Teaching

2002, Winner of the Richard K Lublin Award. Cited for "Ability to engender genuine intellectual excitement, ability to engender curiousity, knowledge of field and ability to communicate that knowledge, organizational skills, creative arrangement of course."

1999, Winner of Outstanding Instructor of Computer Science Award while on sabbatical at University of British Columbia (teaching CPSC 252, a course on Data Structures for approximately 250 Engineering students).

1995, Trinity College, Robert B. Cox Distinguished Teaching in Science Award. Cited for "knowledge of field and ability to communicate it to students, openness to students, skill in organizing courses, commitment to teaching over time."

1994, 1996, 2001, 2002 Nominated by undergraduates for outstanding teaching/faculty award (one of approximately 30 faculty nominated campus-wide each year).

Course and Curriculum Design/Implementation

CPS 53 – Program Analysis and Design I, Fall 1993

In 1993 I led the redesign of the core courses for majors (CPS 06, 08, and 100) to introduce object-oriented programming using C++ and an apprentice style of learning. This led to the development of new courses, described below. This redesign was a significant departure from the C-based courses that had been in place for three years requiring a large-scale change in philosophy as well as significant efforts in developing programming libraries to make the use of C++ feasible for students with no programming experience. This redesign has led to several publications and an NSF grant awarded in December of 1995.

CPS 100E - Program Analysis and Design II, Fall 1995 –

Created a new course for students with programming experience acquired elsewhere (not at Duke), replacing Computer Science 8 and accelerating students into the major. The course reviews material from the end of CPS 6 and then covers material from CPS 100. A formal laboratory component makes this possible. (with S. Rodger)

CPS 108 – Software Design and Implementation, Spring 1995 –

Designed and taught a new course required for all majors (in 1994 formalized the requirement). The course covers advanced object-oriented programming; introducing GUI programming using C++ and Java while emphasizing significant individual and team projects using object-oriented design, analysis, and programming.

In 1995 I established a new software design and engineering component of the curriculum via the course CPS 108. This curricular change led to an NSF CAREER grant for using and developing patterns in teaching software design and introductory programming.

CPS 149S - Problem Solving Seminar, Fall 1994

Created a new seminar course for problem solving, to prepare students for the ACM programming contest. Students worked previous contest problems once a week, and two mini-contests were held. Two teams participated in the regional contest with one team placing first.

CPS 182S – Technical and Social Analysis of Information and the Internet

Designed to meet the needs of Duke's Curriculum 2000. Satisfies, research and writing requirements and science technology and society requirement.

> The development of technical and social standards governing the Internet and Information Technology in general. The role of software as it relates to law, patents, intellectual property, and IETF (Internet Engineering Task Force) standards. Written analysis of issues from a technical perspective with an emphasis on the role of software; but also on how standards relate to social and ethical issues.

In 2002 I designed and had approved CPS 182S, *Technical and Social Analysis of Information and the Internet* a course which R, W, and STS designations as part of Duke's Curriculum 2000 (research, writing, and science, technology and society, respectively). This course led to another, non-major's version of the course in 2008, to publications, and is part of the genesis for the new NSF CS Principles project.

CPS 004G – Programming for Bioinformatics

Designed as one course in a four-course, integrative and interdisciplinary program *The Genome Revolution: Society and Science* for first year students as part of Duke's FOCUS program. The course introduces programming in the context of solving problems from bioinformatics and computational biology.

In 2006 I used this course as a foundation, with work done by Alex Hartemink in our department on a more advanced course, to help spearhead and oversee the process leading to the approval of Duke's first interdepartmental and interdisciplinary minor: Computational Biology and Bioinformatics.

Compsci 82, Technical and Social Foundations of the Internet

In 2008 I used Compsci 182S (see above) as a model for developing Compsci 82, a course without the writing component, but with an Ethical Inquiry (EI) designation. This course was taught in the fall of 2008 to 239 students, the third largest enrollment for a one-section course at Duke.

In 2009 I taught this course to 345 students, the second largest course at Duke and the largest course that does not satisfy a major requirement. In 2010 I again taught the course to 345 students; the course was the largest single-section course taught at Duke.

Compsci 6, Introduction to Computer Science

In 2010 I oversaw the development of a new, introductory course in computer science: Compsci 6, the new description for the course follows.

> Introduction practices and principles of computer science and programming and their impact on and potential to change the world. Algorithmic, problem-solving, and programming techniques in domains such as art, data visualization, mathematics, natural and social sciences. Programming using high-level languages and design techniques emphasizing abstraction, encapsulation, and problem decomposition. Design, implementation, testing, and analysis of algorithms and programs. No previous programming experience required.

This course is intended to appeal to a wider and more diverse audience than our previous version of the course. I developed the infrastructure for the course including developing materials used to teach Python, deciding on the libraries used, and developing the software infrastructure to support the use of Python. I worked with Robert Duvall to deliver the first version of this course in the fall of 2010.

Courses Taught

In the list below, CPS 08/53 corresponds to CS 1 and CPS 100/103 corresponds to CS 2 (courses were renumbered in 1994). CPS 206 is a graduate level programming-languages course. CPS 10 is a comprehensive/breadth first introduction to Computer Science for non-majors.

Date		**Number**	**Title**	**Enrollment**
2010	Spring	Compsci 149s	Problem Solving Seminar	12
		Compsci 100	Program Design and Analysis II	55
		Compsci 182s	Technical and Social Foundations of the Internet	18
	Fall	Compsci 149s	Problem Solving Seminar	8
		Compsci 006	Introduction to Computer Science	79
		Compsci 82	Technical and Social Foundations of the Internet	345
2009	Spring	Compsci 149s	Problem Solving Seminar	11
		Compsci 100	Program Design and Analysis II	29
		Compsci 182s	Technical and Social Foundations of the Internet	20
	Fall	Compsci 149s	Problem Solving Seminar	8
		Compsci 100	Program Design and Analysis II	41
		Compsci 82	Technical and Social Foundations of the Internet	345
2008	Spring	Compsci 149S	Problem Solving Seminar	7+
		Compsci 100	Program Design and Analysis II	33
		Compsci 82S	Technical and Social Foundations of the Internet	23
	Fall	Compsci 82	Technical and Social Foundations of the Internet 239	
		Compsci 100	Program Design and Analysis II	39
		Compsci 149s	Problem Solving Seminar	8+
2007	Spring	Compsci 149S	Problem Solving Seminar	8+
	Fall	Compsci 4G	Genomics Programming (FOCUS)	16
		Compsci 108	Software Design	44
		Compsci 149s	Problem Solving Seminar	5+
2006	Spring	CPS 182s	Technical and Social Analysis of the Internet	20
		CPS 100	Program Design and Analysis II	24
		CPS 149S	Problem Solving Seminar	4+
	Fall	CPS 004g	Introduction to Programming with Genomics (FOCUS)	11
		CPS 100	Program Design and Analysis II	36
		CPS 149S	Problem Solving Seminar	4+
2005	Fall	CPS 004G	Introduction to Programming with Genomics (FOCUS)	17
		CPS 108	Software Design and Implementation	32
		CPS 149S	Problem Solving Seminar	3
2005	Spring	CPS 182s	Technical and Social Analysis of the Internet	15
		CPS 149S	Problem Solving Seminar	4
2004	Fall	CPS 006G	Introduction to Programming with Genomics (FOCUS)	15
		CPS 100	Program Design and Analysis II	35
		CPS 149S	Problem Solving Seminar	7
2004	Spring	CPS 100	Program Design and Analysis II	35
		CPS 149S	Problem Solving Seminar	7
		CPS 108	Software Design and Implementation	45
2003	Fall	CPS 006X	Program Design and Analysis I (honors)	20
		CPS 182S	Technical and Social Analysis of the Internet	35
		CPS 149S	Problem Solving Seminar	7
2003	Spring	CPS 100	Program Design and Analysis II	68
		CPS 108	Software Design and Implementation	99

Date		**Number**	**Title**	**Enrollment**
2002	Fall	CPS 182s	Technical and Social Analysis of the Internet	43
		CPS 149S	Problem Solving Seminar	5
2002	Spring	CPS 100	Program Design and Analysis II	105
2001	Fall	CPS 108	Software Design and Implementation	64
		CPS 06	Program Design adn Analysis I	105
		CPS 149S	Problem Solving Seminar	13
2001	Spring	CPS 108	Software Design and Implementation	124
		CPS 100	Program Design and Analysis II	114
		CPS 189S	CS Education Seminar	3
2000	Fall	CPS 100	Program Design and Analysis II	88
		CPS 149S	Problem Solving Seminar	14
		CPS 189S	CS Education Seminar	6
2000	Spring	CPS 100	Program Design and Analysis II	107
1999	Fall	CPS 108	Software Design and Implementation	115
		CPS 06	Program Design and Analysis I	173
1998	Fall	CS 252 (UBC)	Data Structures,CS2	163
1998	Spring	CPS 108	Software Design and Implementation	94
		CPS 196	Advanced Topics in OO Technology (seminar)	14
1997	Fall	CPS 100	Program Design and Analysis II	61
		CPS 108	Software Design and Implementation	63
		CPS 149S	Problem Solving Seminar	18
1997	Spring	CPS 100	Program Design and Analysis II	72
		CPS 108	Software Design and Implementation	65
1996	Fall	CPS 100E	Program Design and Analysis II	70
		CPS 108	Software Design and Implementation	58
		CPS 149S	Problem Solving Seminar	17
1996	Spring	CPS 100	Program Design and Analysis II	72
		CPS 108	Software Design and Implementation	40
		CPS 208	Software Design and Implementation	15

Date		Number	Title	Enrollment
1995	Fall	CPS 6	Intro. to Program Design/Analysis I	121
		(Team taught with S. Rodger)		
		CPS 100	Program Design and Analysis II	56
		(Team taught with S. Rodger)		
		CPS 100E	Program Design and Analysis II	55
		(Team taught with S. Rodger)		
		CPS 149S	Problem Solving Seminar	14
		(Team taught with S. Rodger)		
	Spring	CPS 108	Software Design and Implementation	38
		CPS 8	Intro. to Program/Design Analysis I	70
1994	Fall	CPS 8	Intro. to Program Design/Analysis I	63
		CPS 100	Program Design and Analysis II	45
	Spring	CPS 8	Intro. to Program Design/Analysis I	67
		CPS 100	Program Design and Analysis II	43
1993	Fall	CPS 8	Intro. to Program Design/Analyis I	39
		CPS 100	Program Design and Analysis II	29
	Spring	CPS 100	Program Design and Analysis II	48
		CPS 206	Programming Languages (graduate)	6
1992	Spring	CPS 10	Fundamentals of Computing	67
1991	Fall	CPS 10	Fundamentals of Computing	135
1987	Fall	CPS 51	Introduction to Programming	90
1987	Spring	CPS 51	Introduction to Programming	101
1986	Fall	CPS 51	Introduction to Programming	196

Thesis Advising

2004: Megan Murphy, *The Uses of Pair Programming in Introductory Computer Science Courses*, thesis for graduation with distinction.

2004: Megan Gessner, *Generation of Spanish Verb Conjugations*, thesis for graduation with distinction.

2002: Donald Onyango, *Comparison of Eduational Tools*, Masters Thesis.

1998: Matthew Kotler, *An interactive CD-based Guide to Duke University*, undergraduate thesis for graduation with Highest Distinction.

1998: Eric Jewart, *Programming in CS 0*, undergraduate thesis for graduation with High Distinction.

1998: Tafawa Kesler, *GOODS: A Design and Class Building Tool*, undergraduate thesis for graduation with Distinction.

1997: Chih-ping Fu, *Towards a Java Bean Building and Using Environment*, Masters Thesis.

Independent Study

Fall 2006, Spring 2007 *Computer and Mathematical Models of Insulin Pathways* Tiffany Chen, graduation with distinction, interdepartmental major, Computer Science and Biology (supervised from Biology by Fred Nijhout).

Fall 2005, Industry/Academic software developement with .NET technologies (with Glaxo-Smith-Kline)

Spring 2005, Web-tools for Russian Vocabulary

Spring 2004, Agile Methods and Programming for Spanish

Fall 2002, Patterns for Networked Games

Fall 2002, Online Grading

Fall 2001, Database-backed web sites: issues and solutions.

Fall 2000, A Framework for an online Calendar System supporting IETF standards

Spring 1998, Developing a CD-based guide to Duke, Advanced OO Design: A Class Browser

Fall 1997, Advanced Object Oriented Design, Interactive Web-based Journaling, Developing a CD-based guide to Duke

Spring 1997, Graphical Debugging

Spring 1997, Distance Learning using a Java Whiteboard

Fall 1996, On-line help by harvesting information with a GUI front end.

Fall 1996, Using C++ in High School Teaching.

Fall 1995, Graphics and Game programming for the Macintosh (6 students).

Summer 1995, Using C++ in High School Teaching

Spring 1995, A GUI/OO interface for air-quality modeling.

Spring 1995, Graphics Programming for the Macintosh.

Spring 1995, Object-Oriented Programming with Smalltalk.

Fall 1994, Implementing an Online Teacher Course Evaluation Book.

Spring 1994, An application-driven approach to foundations of computer science.

Summer 1994, The role of Computer Science for secondary school mathematics teachers.

Summer 1994, Computation Structures and Machine Organization.

Fall 1989, Graphical Display and Manipulation of Data Structures for the Macintosh.

EXHIBIT B: DOCUMENTS AND INFORMATION REVIEWED

a. Oracle's First Amended Complaint

b. Google's Answer to First Amended Complaint and Counterclaims

c. Oracle's Supplemental Responses to Google's Interrogatories, Set No. 1

d. The Android developer website at android.com

e. The Oracle Java websites at java.sun.com and java.oracle.com, including http://java.sun.com/docs/white/platform/javaplatform.doc1.html, http://java.sun.com/docs/books/jls/first_edition/html/index.html, http://java.sun.com/docs/glossary.html

f. Source code, documentation, and discussion boards and blogs for Oracle's implementation of the APIs at issue, including http://download.oracle.com/javase/5/docs/index.html, http://download.oracle.com/javase/1.4/docs/index.html, and http://markmail.org/thread/xwyxemce75vvz33h#query:+page:1+mid:vnipd7bqzs5vxfjw+state:results

g. Source code and documentation for Android's implementation of the APIs at issue, including http://developer.android.com/reference/packages.html and http://android.git.kernel.org/?p=platform/libcore.git;a=history;f=luni/src/test/java/org/apache/harmony/security/tests/java/security/CodeSourceTest.java;hb=a49d9caee4cd74c0d2cf83d79b8ecdc00453dff8

h. Source code and documentation for Apache Harmony's implementation of the APIs at issue, including http://svn.apache.org, http://harmony.apache.org/faq.html and

http://harmony.apache.org/subcomponents/classlibrary/compat.html

i. Source code and documentation for GNU Classpath's implementation of the APIs at issue, including http://www.gnu.org/software/classpath/docs/

j. Wikipedia, *Application programming interface*, http://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=437864024 (as of July 13, 2011, 00:30 GMT)

k. Newton's Telecom Dictionary, 25th Edition

l. Oracle SQL: The Essential Reference," David C. Kreines (2000)

m. C Reference Manual, Dennis Ritchie, 1975, *available at* http://www.cs.bell-labs.com/who/dmr/cman.pdf

n. ZLib Manual, *available at* http://www.zlib.net/manual.html

o. Merriam-Webster Dictionary online

p. Dictionary.com

q. "Revised Report on the Algorithmic Language ALGOL 68", available at http://www.fh-jena.de/~kleine/history/languages/Algol68-RevisedReport.pdf

r. "The C Family of Languages: Interview with Dennis Ritchie, Bjarne Stroustrup, and James Gosling," Java Report, 5(7), July 2000, *available at* http://www.gotw.ca/publications/c_family_interview.htm

s. "The Feeling of Java," James Gosling, *Computer*, Vol. 30, Issue 6, June 1997

t. Mastering Regular Expressions, Jeffrey E. F. Friedl, O'Reilly and Associates, 1997

u. "Programming Techniques: Regular expression search algorithm," Ken Thompson,

Communications of the ACM, Vol. 11, Issue 6, June 1968

v. "SEQUEL: A structured English query language," Proc. ACM SIGFIDET Workshop, May 1974, pp. 249-264

w. JDBC 3.0 Specification, available at http://jcp.org/aboutJava/communityprocess/first/jsr054/jdbc-3_0-pfd-spec.pdf

x. X/Open: Data Management: SQL Call Level Interface (CLI), available at http://pubs.opengroup.org/onlinepubs/009654899/toc.pdf

y. Linux kernel 2.4 source code as available at http://git.kernel.org/?p=linux/kernel/git/stable/linux-2.4.37.y.git, including http://git.kernel.org/?p=linux/kernel/git/stable/linux-2.4.37.y.git;a=blob;f=fs/proc/array.c;h=335226246dcafa18864e87c2f7be68f48a50b924;hb=HEAD

z. Solaris source code and revision history as available at http://cvs.opensolaris.org/source/xref/onnv, including http://cvs.opensolaris.org/source/history/onnv/onnv-gate/usr/src/uts/common/syscall/uucopy.c and http://cvs.opensolaris.org/source/history/onnv/onnv-gate/usr/src/uts/common/brand/; and also as archived at http://hg.genunix.org/onnv-gate.hg/; including http://hg.genunix.org/onnv-gate.hg/rev/4c5722bc28dc

aa. BrandZ documentation at http://hub.opensolaris.org/bin/view/Community+Group+brandz/WebHome and related web pages, including http://hub.opensolaris.org/bin/download/Community+Group+brandz/

WebHome/brandzoverview.pdf and http://hub.opensolaris.org/bin/view/Community+Group+brandz/design

bb. "Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux," 2002, *available at* http://kernel.org/doc/ols/2002/ols2002-pages-479-495.pdf

cc. Futex(2) manual page, *available at* http://www.kernel.org/doc/man-pages/online/pages/man2/futex.2.html

dd. "Excel functions (by category)," http://office.microsoft.com/en-us/excel-help/excel-functions-by-category-HP005204211.aspx

ee. "Calc Functions listed by category," http://wiki.services.openoffice.org/wiki/Documentation/How_Tos/Calc:_Functions_listed_by_category

ff. Visicalc Reference Card, available at http://www.bricklin.com/history/refcard1.htm

gg. "Oracle® Database Application Developer's Guide - Fundamentals," *available at* http://download.oracle.com/docs/cd/B14117_01/appdev.101/b10795/toc.htm

hh. "System R: relational approach to database management," M. M. Astrahan et al; ACM Transactions on Database Systems; Vol. 1, Issue 2, June 1976.

EXHIBIT C: EXCEL AND STAROFFICE SPREADSHEET FUNCTION NAMES

Microsoft Excel 2003	Oracle Open Office Calc (Today)
ABS	ABS
ACCRINT	ACCRINT
ACCRINTM	ACCRINTM
ACOS	ACOS
ACOSH	ACOSH
	ACOT
	ACOTH
ADDRESS	ADDRESS
AMORDEGRC	AMORDEGRC
AMORLINC	AMORLINC
AND	AND
	ARABIC
AREAS	AREAS
ASC	
ASIN	ASIN
ASINH	ASINH
ATAN	ATAN
ATAN2	ATAN2
ATANH	ATANH
AVEDEV	AVEDEV
AVERAGE	AVERAGE
AVERAGEA	AVERAGEA
BAHTTEXT	BAHTTEXT
	BASE
BESSELI	BESSELI
BESSELJ	BESSELJ
BESSELK	BESSELK
BESSELY	BESSELY
BETADIST	BETADIST
BETAINV	BETAINV
BIN2DEC	BIN2DEC
BIN2HEX	BIN2HEX
BIN2OCT	BIN2OCT

BINOMDIST	BINOMDIST
CEILING	CEILING
CELL	CELL
CHAR	CHAR
CHIDIST	CHIDIST
CHIINV	CHIINV
	CHISQDIST
	CHISQINV
CHITEST	CHITEST
CHOOSE	CHOOSE
CLEAN	CLEAN
CODE	CODE
COLUMN	COLUMN
COLUMNS	COLUMNS
COM	
COMBIN	COMBIN
	COMBINA
COMPLEX	COMPLEX
CONCATENATE	CONCATENATE
CONFIDENCE	CONFIDENCE
CONVERT	CONVERT
	CONVERT_ADD
CORREL	CORREL
COS	COS
COSH	COSH
	COT
	COTH
COUNT	COUNT
COUNTA	COUNTA
COUNTBLANK	COUNTBLANK
COUNTIF	COUNTIF
COUPDAYBS	COUPDAYBS
COUPDAYS	COUPDAYS
COUPDAYSNC	COUPDAYSNC
COUPNCD	COUPNCD
COUPNUM	COUPNUM
COUPPCD	COUPPCD

COVAR	COVAR
CRITBINOM	CRITBINOM
CUMIPMT	CUMIPMT
	CUMIPMT_ADD
CUMPRINC	CUMPRINC
	CUMPRINC_ADD
	CURRENT
DATE	DATE
DATEVALUE	DATEVALUE
DAVERAGE	DAVERAGE
DAY	DAY
	DAYS
DAYS360	DAYS360
	DAYSINMONTH
	DAYSINYEAR
DB	DB
DCOUNT	DCOUNT
DCOUNTA	DCOUNTA
DDB	DDB
	DDE
DEC2BIN	DEC2BIN
DEC2HEX	DEC2HEX
DEC2OCT	DEC2OCT
	DECIMAL
DEGREES	DEGREES
DELTA	DELTA
DEVSQ	DEVSQ
DGET	DGET
DISC	DISC
DMAX	DMAX
DMIN	DMIN
DOLLAR	DOLLAR
DOLLARDE	DOLLARDE
DOLLARFR	DOLLARFR
DPRODUCT	DPRODUCT
DSTDEV	DSTDEV
DSTDEVP	DSTDEVP

DSUM	DSUM
DURATION	DURATION
	DURATION_ADD
DVAR	DVAR
DVARP	DVARP
	EASTERSUNDAY
EDATE	EDATE
EFFECT	EFFECT_ADD
	EFFECTIVE
EOMONTH	EOMONTH
ERF	ERF
ERFC	ERFC
ERROR.TYPE	ERRORTYPE
EUROCONVERT	
EVEN	EVEN
EXACT	EXACT
EXP	EXP
EXPONDIST	EXPONDIST
FACT	FACT
FACTDOUBLE	FACTDOUBLE
FALSE	FALSE
FDIST	FDIST
FIND	FIND
FINDB	
FINV	FINV
FISHER	FISHER
FISHERINV	FISHERINV
FIXED	FIXED
FLOOR	FLOOR
FORECAST	FORECAST
	FORMULA
FREQUENCY	FREQUENCY
FTEST	FTEST
FV	FV
FVSCHEDULE	FVSCHEDULE
	GAMMA
GAMMADIST	GAMMADIST

GAMMAINV	GAMMAINV
GAMMALN	GAMMALN
	GAUSS
GCD	GCD
	GCD_ADD
GEOMEAN	GEOMEAN
GESTEP	GESTEP
GETPIVOTDATA	
GROWTH	GROWTH
HARMEAN	HARMEAN
HEX2BIN	HEX2BIN
HEX2DEC	HEX2DEC
HEX2OCT	HEX2OCT
HLOOKUP	HLOOKUP
HOUR	HOUR
HYPERLINK	HYPERLINK
HYPGEOMDIST	HYPGEOMDIST
IF	IF
IMABS	IMABS
IMAGINARY	IMAGINARY
IMARGUMENT	IMARGUMENT
IMCONJUGATE	IMCONJUGATE
IMCOS	IMCOS
IMDIV	IMDIV
IMEXP	IMEXP
IMLN	IMLN
IMLOG10	IMLOG10
IMLOG2	IMLOG2
IMPOWER	IMPOWER
IMPRODUCT	IMPRODUCT
IMREAL	IMREAL
IMSIN	IMSIN
IMSQRT	IMSQRT
IMSUB	IMSUB
IMSUM	IMSUM
INDEX	INDEX
INDIRECT	INDIRECT

INFO	INFO
INT	INT
INTERCEPT	INTERCEPT
INTRATE	INTRATE
IPMT	IPMT
IRR	IRR
ISBLANK	ISBLANK
ISERR	ISERR
ISERROR	ISERROR
ISEVEN	ISEVEN
	ISEVEN_ADD
	ISFORMULA
	ISLEAPYEAR
ISLOGICAL	ISLOGICAL
ISNA	ISNA
ISNONTEXT	ISNONTEXT
ISNUMBER	ISNUMBER
ISODD	ISODD
	ISODD_ADD
ISPMT	ISPMT
ISREF	ISREF
ISTEXT	ISTEXT
IT	
JIS	
KURT	KURT
LARGE	LARGE
LCM	LCM
	LCM_ADD
LEFT	LEFT
LEFTB	
LEN	LEN
LENB	
LINEST	LINEST
LN	LN
LOG	LOG
LOG10	LOG10
LOGEST	LOGEST

LOGINV	LOGINV
LOGNORMDIST	LOGNORMDIST
LOOKUP	LOOKUP
LOWER	LOWER
MATCH	MATCH
MAX	MAX
MAXA	MAXA
MDETERM	MDETERM
MDURATION	MDURATION
MEDIAN	MEDIAN
MID	MID
MIDB	
MIN	MIN
MINA	MINA
MINUTE	MINUTE
MINVERSE	MINVERSE
MIRR	MIRR
MMULT	MMULT
MOD	MOD
MODE	MODE
MONTH	MONTH
	MONTHS
MROUND	MROUND
MULTINOMIAL	MULTINOMIAL
	MUNIT
NA	NA
NEGBINOMDIST	NEGBINOMDIST
NETWORKDAYS	NETWORKDAYS
NOMINAL	NOMINAL
	NOMINAL_ADD
NORMDIST	NORMDIST
NORMINV	NORMINV
NORMSDIST	NORMSDIST
NORMSINV	NORMSINV
NOT	NOT
NOW	NOW
NPER	NPER

NPV	NPV
OCT2BIN	OCT2BIN
OCT2DEC	OCT2DEC
OCT2HEX	OCT2HEX
ODD	ODD
ODDFPRICE	ODDFPRICE
ODDFYIELD	ODDFYIELD
ODDLPRICE	ODDLPRICE
ODDLYIELD	ODDLYIELD
OFFSET	OFFSET
OR	OR
PEARSON	PEARSON
PERCENTILE	PERCENTILE
PERCENTRANK	PERCENTRANK
PERMUT	PERMUT
PHONETIC	
	PERMUTATIONA
	PHI
PI	PI
PMT	PMT
POISSON	POISSON
POWER	POWER
PPMT	PPMT
PRICE	PRICE
PRICEDISC	PRICEDISC
PRICEMAT	PRICEMAT
PROB	PROB
PRODUCT	PRODUCT
PROPER	PROPER
PV	PV
QUARTILE	QUARTILE
QUOTIENT	QUOTIENT
RADIANS	RADIANS
RAND	RAND
RANDBETWEEN	RANDBETWEEN
RANK	RANK
RATE	RATE

RECEIVED	RECEIVED
REPLACE	REPLACE
REPLACEB	
REPT	REPT
RIGHT	RIGHT
RIGHTB	
ROMAN	ROMAN
ROUND	ROUND
ROUNDDOWN	ROUNDDOWN
ROUNDUP	ROUNDUP
ROW	ROW
ROWS	ROWS
	RRI
RSQ	RSQ
RTD	
SEARCH	SEARCH
SEARCHB	
SECOND	SECOND
SERIESSUM	SERIESSUM
	SHEET
	SHEETS
SIGN	SIGN
SIN	SIN
SINH	SINH
SKEW	SKEW
SLN	SLN
SLOPE	SLOPE
SMALL	SMALL
SQL.REQUEST	
SQRT	SQRT
SQRTPI	SQRTPI
STANDARDIZE	STANDARDIZE
STDEV	STDEV
STDEVA	STDEVA
STDEVP	STDEVP
STDEVPA	STDEVPA
STEYX	STEYX

	STYLE
SUBSTITUTE	SUBSTITUTE
SUBTOTAL	SUBTOTAL
SUM	SUM
SUMIF	SUMIF
SUMPRODUCT	SUMPRODUCT
SUMSQ	SUMSQ
SUMX2MY2	SUMX2MY2
SUMX2PY2	SUMX2PY2
SUMXMY2	SUMXMY2
SYD	SYD
TAN	TAN
TANH	TANH
TBILLEQ	TBILLEQ
TBILLPRICE	TBILLPRICE
TBILLYIELD	TBILLYIELD
TDIST	TDIST
TEXT	TEXT
TIME	TIME
TIMEVALUE	TIMEVALUE
TINV	TINV
TODAY	TODAY
TRANSPOSE	TRANSPOSE
TREND	TREND
TRIM	TRIM
TRIMMEAN	TRIMMEAN
TRUE	TRUE
TRUNC	TRUNC
TTEST	TTEST
TYPE	TYPE
UPPER	UPPER
VALUE	VALUE
VAR	VAR
VARA	VARA
VARP	VARP
VARPA	VARPA
VDB	VDB

VLOOKUP	VLOOKUP
WEEKDAY	WEEKDAY
WEEKNUM	WEEKNUM
	WEEKNUM_ADD
	WEEKS
	WEEKSINYEAR
WEIBULL	WEIBULL
WORKDAY	WORKDAY
XIRR	XIRR
XNPV	XNPV
YEAR	YEAR
YEARFRAC	YEARFRAC
	YEARS
YIELD	YIELD
YIELDDISC	YIELDDISC
YIELDMAT	YIELDMAT
ZTEST	ZTEST

EXHIBIT D: LX_BRAND SYSCALL TABLE

From lx_brand/common/lx_brand.c:

```
static struct lx_sysent sysents[] = {

        {"nosys",       NULL,           NOSYS_NULL,     0},     /*  0 */
        {"exit",        lx_exit,        0,              1},     /*  1 */
        {"fork",        lx_fork,        0,              0},     /*  2 */
        {"read",        lx_read,        0,              3},     /*  3 */
        {"write",       write,          SYS_PASSTHRU,   3},     /*  4 */
        {"open",        lx_open,        0,              3},     /*  5 */
        {"close",       close,          SYS_PASSTHRU,   1},     /*  6 */
        {"waitpid",     lx_waitpid,     0,              3},     /*  7 */
        {"creat",       creat,          SYS_PASSTHRU,   2},     /*  8 */
        {"link",        lx_link,        0,              2},     /*  9 */
        {"unlink",      lx_unlink,      0,              1},     /* 10 */
        {"execve",      lx_execve,      0,              3},     /* 11 */
        {"chdir",       chdir,          SYS_PASSTHRU,   1},     /* 12 */
        {"time",        lx_time,        0,              1},     /* 13 */
        {"mknod",       lx_mknod,       0,              3},     /* 14 */
        {"chmod",       lx_chmod,       0,              2},     /* 15 */
        {"lchown16",    lx_lchown16,    0,              3},     /* 16 */
        {"break",       NULL,           NOSYS_OBSOLETE, 0},     /* 17 */
        {"stat",        NULL,           NOSYS_OBSOLETE, 0},     /* 18 */
        {"lseek",       lx_lseek,       0,              3},     /* 19 */
        {"getpid",      lx_getpid,      0,              0},     /* 20 */
        {"mount",       lx_mount,       0,              5},     /* 21 */
```

```
{"umount",      lx_umount,      0,               1},     /* 22 */
{"setuid16",    lx_setuid16,    0,               1},     /* 23 */
{"getuid16",    lx_getuid16,    0,               0},     /* 24 */
{"stime",       stime,          SYS_PASSTHRU,    1},     /* 25 */
{"ptrace",      lx_ptrace,      0,               4},     /* 26 */
{"alarm",       (int (*)())alarm, SYS_PASSTHRU, 1},     /* 27 */
{"fstat",       NULL,           NOSYS_OBSOLETE, 0},     /* 28 */
{"pause",       pause,          SYS_PASSTHRU,    0},     /* 29 */
{"utime",       lx_utime,       0,               2},     /* 30 */
{"stty",        NULL,           NOSYS_OBSOLETE, 0},     /* 31 */
{"gtty",        NULL,           NOSYS_OBSOLETE, 0},     /* 32 */
{"access",      access,         SYS_PASSTHRU,    2},     /* 33 */
{"nice",        nice,           SYS_PASSTHRU,    1},     /* 34 */
{"ftime",       NULL,           NOSYS_OBSOLETE, 0},     /* 35 */
{"sync",        lx_sync,        0,               0},     /* 36 */
{"kill",        lx_kill,        0,               2},     /* 37 */
{"rename",      lx_rename,      0,               2},     /* 38 */
{"mkdir",       mkdir,          SYS_PASSTHRU,    2},     /* 39 */
{"rmdir",       lx_rmdir,       0,               1},     /* 40 */
{"dup",         dup,            SYS_PASSTHRU,    1},     /* 41 */
{"pipe",        lx_pipe,        0,               1},     /* 42 */
{"times",       lx_times,       0,               1},     /* 43 */
{"prof",        NULL,           NOSYS_OBSOLETE, 0},     /* 44 */
{"brk",         lx_brk,         0,               1},     /* 45 */
{"setgid16",    lx_setgid16,    0,               1},     /* 46 */
{"getgid16",    lx_getgid16,    0,               0},     /* 47 */
```

```
{"signal",      lx_signal,      0,              2},     /* 48 */
{"geteuid16",   lx_geteuid16,   0,              0},     /* 49 */
{"getegid16",   lx_getegid16,   0,              0},     /* 50 */
{"acct",        NULL,           NOSYS_NO_EQUIV, 0},     /* 51 */
{"umount2",     lx_umount2,     0,              2},     /* 52 */
{"lock",        NULL,           NOSYS_OBSOLETE, 0},     /* 53 */
{"ioctl",       lx_ioctl,       0,              3},     /* 54 */
{"fcntl",       lx_fcntl,       0,              3},     /* 55 */
{"mpx",         NULL,           NOSYS_OBSOLETE, 0},     /* 56 */
{"setpgid",     lx_setpgid,     0,              2},     /* 57 */
{"ulimit",      NULL,           NOSYS_OBSOLETE, 0},     /* 58 */
{"olduname",    NULL,           NOSYS_OBSOLETE, 0},     /* 59 */
{"umask",       (int (*)())umask, SYS_PASSTHRU, 1},     /* 60 */
{"chroot",      chroot,         SYS_PASSTHRU,   1},     /* 61 */
{"ustat",       lx_ustat,       0,              2},     /* 62 */
{"dup2",        lx_dup2,        0,              2},     /* 63 */
{"getppid",     lx_getppid,     0,              0},     /* 64 */
{"getpgrp",     lx_getpgrp,     0,              0},     /* 65 */
{"setsid",      lx_setsid,      0,              0},     /* 66 */
{"sigaction",   lx_sigaction,   0,              3},     /* 67 */
{"sgetmask",    NULL,           NOSYS_OBSOLETE, 0},     /* 68 */
{"ssetmask",    NULL,           NOSYS_OBSOLETE, 0},     /* 69 */
{"setreuid16",  lx_setreuid16,  0,              2},     /* 70 */
{"setregid16",  lx_setregid16,  0,              2},     /* 71 */
{"sigsuspend",  lx_sigsuspend,  0,              1},     /* 72 */
{"sigpending",  lx_sigpending,  0,              1},     /* 73 */
```

```
{"sethostname", lx_sethostname, 0,              2},     /* 74 */
{"setrlimit",   lx_setrlimit,   0,              2},     /* 75 */
{"getrlimit",   lx_oldgetrlimit, 0,             2},     /* 76 */
{"getrusage",   lx_getrusage,   0,              2},     /* 77 */
{"gettimeofday", lx_gettimeofday, 0,            2},     /* 78 */
{"settimeofday", lx_settimeofday, 0,            2},     /* 79 */
{"getgroups16", lx_getgroups16, 0,              2},     /* 80 */
{"setgroups16", lx_setgroups16, 0,              2},     /* 81 */
{"select",      NULL,           NOSYS_OBSOLETE, 0},     /* 82 */
{"symlink",     symlink,        SYS_PASSTHRU,   2},     /* 83 */
{"oldlstat",    NULL,           NOSYS_OBSOLETE, 0},     /* 84 */
{"readlink",    readlink,       SYS_PASSTHRU,   3},     /* 85 */
{"uselib",      NULL,           NOSYS_KERNEL,   0},     /* 86 */
{"swapon",      NULL,           NOSYS_KERNEL,   0},     /* 87 */
{"reboot",      lx_reboot,      0,              4},     /* 88 */
{"readdir",     lx_readdir,     0,              3},     /* 89 */
{"mmap",        lx_mmap,        0,              6},     /* 90 */
{"munmap",      munmap,         SYS_PASSTHRU,   2},     /* 91 */
{"truncate",    lx_truncate,    0,              2},     /* 92 */
{"ftruncate",   lx_ftruncate,   0,              2},     /* 93 */
{"fchmod",      fchmod,         SYS_PASSTHRU,   2},     /* 94 */
{"fchown16",    lx_fchown16,    0,              3},     /* 95 */
{"getpriority", lx_getpriority, 0,              2},     /* 96 */
{"setpriority", lx_setpriority, 0,              3},     /* 97 */
{"profil",      NULL,           NOSYS_NO_EQUIV, 0},     /* 98 */
{"statfs",      lx_statfs,      0,              2},     /* 99 */
```

```
{"fstatfs",     lx_fstatfs,     0,              2},     /* 100 */
{"ioperm",      NULL,           NOSYS_NO_EQUIV, 0},     /* 101 */
{"socketcall",  lx_socketcall,  0,              2},     /* 102 */
{"syslog",      NULL,           NOSYS_KERNEL,   0},     /* 103 */
{"setitimer",   lx_setitimer,   0,              3},     /* 104 */
{"getitimer",   getitimer,      SYS_PASSTHRU,   2},     /* 105 */
{"stat",        lx_stat,        0,              2},     /* 106 */
{"lstat",       lx_lstat,       0,              2},     /* 107 */
{"fstat",       lx_fstat,       0,              2},     /* 108 */
{"uname",       NULL,           NOSYS_OBSOLETE, 0},     /* 109 */
{"oldiopl",     NULL,           NOSYS_NO_EQUIV, 0},     /* 110 */
{"vhangup",     lx_vhangup,     0,              0},     /* 111 */
{"idle",        NULL,           NOSYS_NO_EQUIV, 0},     /* 112 */
{"vm86old",     NULL,           NOSYS_OBSOLETE, 0},     /* 113 */
{"wait4",       lx_wait4,       0,              4},     /* 114 */
{"swapoff",     NULL,           NOSYS_KERNEL,   0},     /* 115 */
{"sysinfo",     lx_sysinfo,     0,              1},     /* 116 */
{"ipc",         lx_ipc,         0,              5},     /* 117 */
{"fsync",       lx_fsync,       0,              1},     /* 118 */
{"sigreturn",   lx_sigreturn,   0,              1},     /* 119 */
{"clone",       lx_clone,       0,              5},     /* 120 */
{"setdomainname", lx_setdomainname, 0,          2},     /* 121 */
{"uname",       lx_uname,       0,              1},     /* 122 */
{"modify_ldt",  lx_modify_ldt,  0,              3},     /* 123 */
{"adjtimex",    lx_adjtimex,    0,              1},     /* 124 */
{"mprotect",    lx_mprotect,    0,              3},     /* 125 */
```

```
{"sigprocmask", lx_sigprocmask, 0,              3},     /* 126 */
{"create_module", NULL,         NOSYS_KERNEL,   0},     /* 127 */
{"init_module", NULL,           NOSYS_KERNEL,   0},     /* 128 */
{"delete_module", NULL,         NOSYS_KERNEL,   0},     /* 129 */
{"get_kernel_syms", NULL,       NOSYS_KERNEL,   0},     /* 130 */
{"quotactl",    NULL,           NOSYS_KERNEL,   0},     /* 131 */
{"getpgid",     lx_getpgid,     0,              1},     /* 132 */
{"fchdir",      fchdir,         SYS_PASSTHRU,   1},     /* 133 */
{"bdflush",     NULL,           NOSYS_KERNEL,   0},     /* 134 */
{"sysfs",       lx_sysfs,       0,              3},     /* 135 */
{"personality", lx_personality, 0,              1},     /* 136 */
{"afs_syscall", NULL,           NOSYS_KERNEL,   0},     /* 137 */
{"setfsuid16",  lx_setfsuid16,  0,              1},     /* 138 */
{"setfsgid16",  lx_setfsgid16,  0,              1},     /* 139 */
{"llseek",      lx_llseek,      0,              5},     /* 140 */
{"getdents",    getdents,       SYS_PASSTHRU,   3},     /* 141 */
{"select",      lx_select,      0,              5},     /* 142 */
{"flock",       lx_flock,       0,              2},     /* 143 */
{"msync",       lx_msync,       0,              3},     /* 144 */
{"readv",       lx_readv,       0,              3},     /* 145 */
{"writev",      lx_writev,      0,              3},     /* 146 */
{"getsid",      lx_getsid,      0,              1},     /* 147 */
{"fdatasync",   lx_fdatasync,   0,              1},     /* 148 */
{"sysctl",      lx_sysctl,      0,              1},     /* 149 */
{"mlock",       lx_mlock,       0,              2},     /* 150 */
{"munlock",     lx_munlock,     0,              2},     /* 151 */
```

```
        {"mlockall",     lx_mlockall,    0,              1},      /* 152 */
        {"munlockall",   lx_munlockall,  0,              0},      /* 153 */
        {"sched_setparam", lx_sched_setparam,   0,       2},      /* 154 */
        {"sched_getparam", lx_sched_getparam,   0,       2},      /* 155 */
        {"sched_setscheduler", lx_sched_setscheduler, 0, 3},      /* 156 */
        {"sched_getscheduler", lx_sched_getscheduler, 0, 1},      /* 157 */
        {"sched_yield", (int (*)())yield, SYS_PASSTHRU, 0},       /* 158 */
        {"sched_get_priority_max", lx_sched_get_priority_max, 0, 1}, /* 159
*/
        {"sched_get_priority_min", lx_sched_get_priority_min, 0, 1}, /* 160
*/
        {"sched_rr_get_interval", lx_sched_rr_get_interval, 0,  2},  /* 161
*/
        {"nanosleep",    nanosleep,      SYS_PASSTHRU,   2},      /* 162 */
        {"mremap",       NULL,           NOSYS_NO_EQUIV, 0},      /* 163 */
        {"setresuid16", lx_setresuid16, 0,               3},      /* 164 */
        {"getresuid16", lx_getresuid16, 0,               3},      /* 165 */
        {"vm86",         NULL,           NOSYS_NO_EQUIV, 0},      /* 166 */
        {"query_module", lx_query_module, NOSYS_KERNEL, 5},       /* 167 */
        {"poll",         lx_poll,        0,              3},      /* 168 */
        {"nfsservctl",   NULL,           NOSYS_KERNEL,   0},      /* 169 */
        {"setresgid16", lx_setresgid16, 0,               3},      /* 170 */
        {"getresgid16", lx_getresgid16, 0,               3},      /* 171 */
        {"prctl",        NULL,           NOSYS_UNDOC,    0},      /* 172 */
        {"rt_sigreturn", lx_rt_sigreturn, 0,             0},      /* 173 */
        {"rt_sigaction", lx_rt_sigaction, 0,             4},      /* 174 */
```

```
{"rt_sigprocmask", lx_rt_sigprocmask, 0,          4},     /* 175 */
{"rt_sigpending", lx_rt_sigpending, 0,            2},     /* 176 */
{"rt_sigtimedwait", lx_rt_sigtimedwait, 0,        4},     /* 177 */
{"sigqueueinfo", NULL,           NOSYS_UNDOC,     0},     /* 178 */
{"rt_sigsuspend", lx_rt_sigsuspend, 0,            2},     /* 179 */
{"pread64",      lx_pread64,     0,               5},     /* 180 */
{"pwrite64",     lx_pwrite64,    0,               5},     /* 181 */
{"chown16",      lx_chown16,     0,               3},     /* 182 */
{"getcwd",       lx_getcwd,      0,               2},     /* 183 */
{"capget",       NULL,           NOSYS_NO_EQUIV,  0},     /* 184 */
{"capset",       NULL,           NOSYS_NO_EQUIV,  0},     /* 185 */
{"sigaltstack",  lx_sigaltstack, 0,               2},     /* 186 */
{"sendfile",     lx_sendfile,    0,               4},     /* 187 */
{"getpmsg",      NULL,           NOSYS_OBSOLETE,  0},     /* 188 */
{"putpmsg",      NULL,           NOSYS_OBSOLETE,  0},     /* 189 */
{"vfork",        lx_vfork,       0,               0},     /* 190 */
{"getrlimit",    lx_getrlimit,   0,               2},     /* 191 */
{"mmap2",        lx_mmap2,       EBP_HAS_ARG6,    6},     /* 192 */
{"truncate64",   lx_truncate64,  0,               3},     /* 193 */
{"ftruncate64",  lx_ftruncate64, 0,               3},     /* 194 */
{"stat64",       lx_stat64,      0,               2},     /* 195 */
{"lstat64",      lx_lstat64,     0,               2},     /* 196 */
{"fstat64",      lx_fstat64,     0,               2},     /* 197 */
{"lchown",       lchown,         SYS_PASSTHRU,    3},     /* 198 */
{"getuid",       (int (*)())getuid, SYS_PASSTHRU, 0},     /* 199 */
{"getgid",       (int (*)())getgid, SYS_PASSTHRU, 0},     /* 200 */
```

```
{"geteuid",     lx_geteuid,     0,              0},     /* 201 */
{"getegid",     lx_getegid,     0,              0},     /* 202 */
{"setreuid",    setreuid,       SYS_PASSTHRU,   0},     /* 203 */
{"setregid",    setregid,       SYS_PASSTHRU,   0},     /* 204 */
{"getgroups",   getgroups,      SYS_PASSTHRU,   2},     /* 205 */
{"setgroups",   lx_setgroups,   0,              2},     /* 206 */
{"fchown",      lx_fchown,      0,              3},     /* 207 */
{"setresuid",   lx_setresuid,   0,              3},     /* 208 */
{"getresuid",   lx_getresuid,   0,              3},     /* 209 */
{"setresgid",   lx_setresgid,   0,              3},     /* 210 */
{"getresgid",   lx_getresgid,   0,              3},     /* 211 */
{"chown",       lx_chown,       0,              3},     /* 212 */
{"setuid",      setuid,         SYS_PASSTHRU,   1},     /* 213 */
{"setgid",      setgid,         SYS_PASSTHRU,   1},     /* 214 */
{"setfsuid",    lx_setfsuid,    0,              1},     /* 215 */
{"setfsgid",    lx_setfsgid,    0,              1},     /* 216 */
{"pivot_root",  NULL,           NOSYS_KERNEL,   0},     /* 217 */
{"mincore",     mincore,        SYS_PASSTHRU,   3},     /* 218 */
{"madvise",     lx_madvise,     0,              3},     /* 219 */
{"getdents64",  lx_getdents64,  0,              3},     /* 220 */
{"fcntl64",     lx_fcntl64,     0,              3},     /* 221 */
{"tux",         NULL,           NOSYS_NO_EQUIV, 0},     /* 222 */
{"security",    NULL,           NOSYS_NO_EQUIV, 0},     /* 223 */
{"gettid",      lx_gettid,      0,              0},     /* 224 */
{"readahead",   NULL,           NOSYS_NO_EQUIV, 0},     /* 225 */
{"setxattr",    NULL,           NOSYS_NO_EQUIV, 0},     /* 226 */
```

```
{"lsetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 227 */
{"fsetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 228 */
{"getxattr",    NULL,           NOSYS_NO_EQUIV, 0},     /* 229 */
{"lgetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 230 */
{"fgetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 231 */
{"listxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 232 */
{"llistxattr",  NULL,           NOSYS_NO_EQUIV, 0},     /* 233 */
{"flistxattr",  NULL,           NOSYS_NO_EQUIV, 0},     /* 234 */
{"removexattr", NULL,           NOSYS_NO_EQUIV, 0},     /* 235 */
{"lremovexattr", NULL,          NOSYS_NO_EQUIV, 0},     /* 236 */
{"fremovexattr", NULL,          NOSYS_NO_EQUIV, 0},     /* 237 */
{"tkill",       lx_tkill,       0,              2},     /* 238 */
{"sendfile64",  lx_sendfile64,  0,              4},     /* 239 */
{"futex",       lx_futex,       EBP_HAS_ARG6,   6},     /* 240 */
{"sched_setaffinity",   lx_sched_setaffinity,   0, 3},  /* 241 */
{"sched_getaffinity",   lx_sched_getaffinity,   0, 3},  /* 242 */
{"set_thread_area", lx_set_thread_area, 0,      1},     /* 243 */
{"get_thread_area", lx_get_thread_area, 0,      1},     /* 244 */
{"io_setup",    NULL,           NOSYS_NO_EQUIV, 0},     /* 245 */
{"io_destroy",  NULL,           NOSYS_NO_EQUIV, 0},     /* 246 */
{"io_getevents", NULL,          NOSYS_NO_EQUIV, 0},     /* 247 */
{"io_submit",   NULL,           NOSYS_NO_EQUIV, 0},     /* 248 */
{"io_cancel",   NULL,           NOSYS_NO_EQUIV, 0},     /* 249 */
{"fadvise64",   NULL,           NOSYS_UNDOC,    0},     /* 250 */
{"nosys",       NULL,           0,              0},     /* 251 */
{"group_exit",  lx_group_exit,  0,              1},     /* 252 */
```

```
{"lookup_dcookie", NULL,        NOSYS_NO_EQUIV, 0},     /* 253 */
{"epoll_create", NULL,          NOSYS_NO_EQUIV, 0},     /* 254 */
{"epoll_ctl",    NULL,          NOSYS_NO_EQUIV, 0},     /* 255 */
{"epoll_wait",   NULL,          NOSYS_NO_EQUIV, 0},     /* 256 */
{"remap_file_pages", NULL,      NOSYS_NO_EQUIV, 0},     /* 257 */
{"set_tid_address", lx_set_tid_address, 0,      1},     /* 258 */
{"timer_create", NULL,          NOSYS_UNDOC,    0},     /* 259 */
{"timer_settime", NULL,         NOSYS_UNDOC,    0},     /* 260 */
{"timer_gettime", NULL,         NOSYS_UNDOC,    0},     /* 261 */
{"timer_getoverrun", NULL,      NOSYS_UNDOC,    0},     /* 262 */
{"timer_delete", NULL,          NOSYS_UNDOC,    0},     /* 263 */
{"clock_settime", lx_clock_settime,     0,      2},     /* 264 */
{"clock_gettime", lx_clock_gettime,     0,      2},     /* 265 */
{"clock_getres", lx_clock_getres,       0,      2},     /* 266 */
{"clock_nanosleep", lx_clock_nanosleep, 0,      4},     /* 267 */
{"statfs64",     lx_statfs64,   0,              2},     /* 268 */
{"fstatfs64",    lx_fstatfs64,  0,              2},     /* 269 */
{"tgkill",       lx_tgkill,     0,              3},     /* 270 */
/* The following system calls only exist in kernel 2.6 and greater */
{"utimes",       utimes,        SYS_PASSTHRU,   2},     /* 271 */
{"fadvise64_64", NULL,          NOSYS_NULL,     0},     /* 272 */
{"vserver",      NULL,          NOSYS_NULL,     0},     /* 273 */
{"mbind",        NULL,          NOSYS_NULL,     0},     /* 274 */
{"get_mempolicy", NULL,         NOSYS_NULL,     0},     /* 275 */
{"set_mempolicy", NULL,         NOSYS_NULL,     0},     /* 276 */
{"mq_open",      NULL,          NOSYS_NULL,     0},     /* 277 */
```

```
{"mq_unlink",    NULL,           NOSYS_NULL,     0},     /* 278 */
{"mq_timedsend", NULL,           NOSYS_NULL,     0},     /* 279 *
{"mq_timedreceive", NULL,        NOSYS_NULL,     0},     /* 280 */
{"mq_notify",    NULL,           NOSYS_NULL,     0},     /* 281 */
{"mq_getsetattr", NULL,          NOSYS_NULL,     0},     /* 282 */
{"kexec_load",   NULL,           NOSYS_NULL,     0},     /* 283 */
{"waitid",       lx_waitid,      0,              4},     /* 284 */
{"sys_setaltroot", NULL,         NOSYS_NULL,     0},     /* 285 */
{"add_key",      NULL,           NOSYS_NULL,     0},     /* 286 */
{"request_key", NULL,            NOSYS_NULL,     0},     /* 287 */
{"keyctl",       NULL,           NOSYS_NULL,     0},     /* 288 */
{"ioprio_set",   NULL,           NOSYS_NULL,     0},     /* 289 */
{"ioprio_get",   NULL,           NOSYS_NULL,     0},     /* 290 */
{"inotify_init", NULL,           NOSYS_NULL,     0},     /* 291 */
{"inotify_add_watch", NULL,      NOSYS_NULL,     0},     /* 292 */
{"inotify_rm_watch", NULL,       NOSYS_NULL,     0},     /* 293 */
{"migrate_pages", NULL,          NOSYS_NULL,     0},     /* 294 */
{"openat",       lx_openat,      0,              4},     /* 295 */
{"mkdirat",      lx_mkdirat,     0,              3},     /* 296 */
{"mknodat",      lx_mknodat,     0,              4},     /* 297 */
{"fchownat",     lx_fchownat,    0,              5},     /* 298 */
{"futimesat",    lx_futimesat,   0,              3},     /* 299 */
{"fstatat64",    lx_fstatat64,   0,              4},     /* 300 */
{"unlinkat",     lx_unlinkat,    0,              3},     /* 301 */
{"renameat",     lx_renameat,    0,              4},     /* 302 */
{"linkat",       lx_linkat,      0,              5},     /* 303 */
```

```
        {"symlinkat",    lx_symlinkat,    0,            3},      /* 304 */
        {"readlinkat",   lx_readlinkat,   0,            4},      /* 305 */
        {"fchmodat",     lx_fchmodat,     0,            4},      /* 306 */
        {"faccessat",    lx_faccessat,    0,            4},      /* 307 */
        {"pselect6",     NULL,            NOSYS_NULL,   0},      /* 308 */
        {"ppoll",        NULL,            NOSYS_NULL,   0},      /* 309 */
        {"unshare",      NULL,            NOSYS_NULL,   0},      /* 310 */
        {"set_robust_list", NULL,         NOSYS_NULL,   0},      /* 311 */
        {"get_robust_list", NULL,         NOSYS_NULL,   0},      /* 312 */
        {"splice",       NULL,            NOSYS_NULL,   0},      /* 313 */
        {"sync_file_range", NULL,         NOSYS_NULL,   0},      /* 314 */
        {"tee",          NULL,            NOSYS_NULL,   0},      /* 315 */
        {"vmsplice",     NULL,            NOSYS_NULL,   0},      /* 316 */
        {"move_pages",   NULL,            NOSYS_NULL,   0},      /* 317 */
};
```

EXHIBIT E: SOURCE CODE FOR SLOCCOUNTER.PY AND SLOCCOUNTERTOTAL.PY

```
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''

import os,collections

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aset = set()
apack = {}


jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jset = set()
jpack = {}

public_ids = ["public class",
              "public abstract class",
              "public interface",
              "protected class",
              "protected",
              "public"]

def do_one(onepath,cset,cpack):
    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package-info.java"):
        return True
    f = open(onepath)

    pcount = 0
    first = True
    public = False
    for line in f:

        line = line.strip()

        if first and line.startswith("public class "):
            #print "class",onepath,line
            base = os.path.basename(onepath)
            cset.add(base)
            cpack[base] = onepath
            break

    return public

def topcount(basepath,packname,cset,cpack):
    parts = packname.split(".")
    pathize = '/'.join(parts)
    packagepath = os.path.join(basepath,pathize)
    for top in os.listdir(packagepath):
        top_path = os.path.join(packagepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)
            pass
        else:
            c = do_one(top_path,cset,cpack)
            if not c:
                #print "no public",top_path,top
                pass
            #print "%s has %d public" % (top_path,c)
def revs(s):
    return s[::-1]

def analyze():

    jpath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
```



```
    packages = ["java.awt.font",
                "java.beans",
                "java.io",
                "java.lang",
                "java.lang.annotation",
                "java.lang.ref",
                "java.lang.reflect",
                "java.math",
                "java.net",
                "java.nio",
                "java.nio.channels",
                "java.nio.channels.spi",
                "java.nio.charset",
                "java.nio.charset.spi",
                "java.security",
                "java.security.acl",
                "java.security.cert",
                "java.security.interfaces",
                "java.security.spec",
                "java.sql",
                "java.text",
                "java.util",
                #"java.util.concurrent",
                #"java.util.concurrrent.atomic",
                #"java.util.concurrent.locks",
                "java.util.jar",
                "java.util.logging",
                "java.util.prefs",
                "java.util.regex",
                "java.util.zip",
                "javax.crypto",
                "javax.crypto.interfaces",
                "javax.crypto.spec",
                "javax.net",
                "javax.net.ssl",
                "javax.security.auth",
                "javax.security.auth.callback",
                "javax.security.auth.login",
                "javax.security.auth.x500",
                "javax.security.cert",
                "javax.sql",
                "javax.xml",
                "javax.xml.datatype",
                "javax.xml.namespace",
                "javax.xml.parsers",
                "javax.xml.transform",
                "javax.xml.transform.dom",
                "javax.xml.transform.sax",
                "javax.xml.transform.stream",
                "javax.xml.validation",
                "javax.xml.xpath"
                ]

    for pack in packages:
        topcount(apath,pack,aset,apack)
        topcount(jpath,pack,jset,jpack)


    names = sorted(jset,key=revs)
    for i,name in enumerate(names):
        print "%d\t%s" % (i,name)

    for i,name in enumerate(names):
        print "%d %s" % (i,jpack[name])


if __name__ == "__main__":
    analyze()
```

```python
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''

import os,collections

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aset = set()

jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jset = set()


public_ids = ["public class",
              "public abstract class",
              "public interface",
              "protected class",
              "protected",
              "public"]

def do_one(onepath,cdict,perclass,cset):
    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package-info.java"):
        return True
    f = open(onepath)
    pcount = 0
    first = True
    public = False
    for line in f:

        line = line.strip()

        if first and line.startswith("class "):
            #print "class",onepath,line
            base = os.path.basename(onepath)
            cset.add(base)

        pfound = False
        for pub in public_ids:

            if line.startswith(pub):
                if first:
                    first = False
                    if line.find("public") >= 0 or line.find("protected") >= 0:
                        public = True
                    else:
                        print "big problem",onepath,pub,line
                if line.find("protected") < 0:
                    pcount += 1
                cdict[pub] += 1
                pfound = True
                if line.find("class") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
                elif line.find("interface") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
                break

    f.close()
    perclass[pcount] += 1
    if pcount == 0:
        #print "%s = %d" % (onepath,pcount)
        pass
    return public

def topcount(basepath,packname,cdict,perclass,cset):
    parts = packname.split(".")
```



```python
    pathize = '/'.join(parts)
    packagepath = os.path.join(basepath,pathize)
    for top in os.listdir(packagepath):
        top_path = os.path.join(packagepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)
            pass
        else:
            c = do_one(top_path,cdict,perclass,cset)
            if not c:
                #print "no public",top_path,top
                pass
            #print "%s has %d public" % (top_path,c)

def report(cdict,perclass):
    ctotal = 0
    for key in cdict:
        if key.find("public") < 0:
            continue
        print "%s occurrences = %d" % (key,cdict[key])
        if key.find("class") >= 0 or key.find("interface") >= 0:
            ctotal += cdict[key]
    print "----"
    print "public class/interface total = %d" % (ctotal)

    ctotal = 0
    for key in cdict:
        if key.find("protected") < 0:
            continue
        print "%s occurrences = %d" % (key,cdict[key])
        if key.find("class") >= 0 or key.find("interface") >= 0:
            ctotal += cdict[key]
    print "----"
    print "protected class/interface total = %d" % (ctotal)

    print "per class method counts"
    print "# methods\t#classes"
    total = 0
    levels = collections.defaultdict(int)
    levlist = [0,1,6,11,16,21,51,101,100001]
    for method_count in sorted(perclass.keys()):
        print "%d\t%d" % (method_count,perclass[method_count])
        total += method_count*perclass[method_count]
        for lev in xrange(1,len(levlist)):
            if levlist[lev-1] <= method_count < levlist[lev]:
                levels[lev] += perclass[method_count]
    print "-----"
    print "total methods = %d" % (total)
    print "\n---summary--"
    total = 0
    for lev in xrange(1,len(levlist)):
        print "perclass from %d to %d = %d" % (levlist[lev-1],levlist[lev]-1,levels[lev])
        total += levels[lev]
    print "total = %d" % (total)

def analyze():

    apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
    javapath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    gnupath = "/Users/ola/expert/google/source-gnu/classpath-0.98"

    packages = ["java.awt.font",
                "java.beans",
                "java.io",
                "java.lang",
                "java.lang.annotation",
                "java.lang.ref",
                "java.lang.reflect",
                "java.math",
                "java.net",
                "java.nio",
                "java.nio.channels",
```

```
                "java.nio.channels.spi",
                "java.nio.charset",
                "java.nio.charset.spi",
                "java.security",
                "java.security.acl",
                "java.security.cert",
                "java.security.interfaces",
                "java.security.spec",
                "java.sql",
                "java.text",
                "java.util",
                #"java.util.concurrent",
                #"java.util.concurrrent.atomic",
                #"java.util.concurrent.locks",
                "java.util.jar",
                "java.util.logging",
                "java.util.prefs",
                "java.util.regex",
                "java.util.zip",
                "javax.crypto",
                "javax.crypto.interfaces",
                "javax.crypto.spec",
                "javax.net",
                "javax.net.ssl",
                "javax.security.auth",
                "javax.security.auth.callback",
                "javax.security.auth.login",
                "javax.security.auth.x500",
                "javax.security.cert",
                "javax.sql",
                "javax.xml",
                "javax.xml.datatype",
                "javax.xml.namespace",
                "javax.xml.parsers",
                "javax.xml.transform",
                "javax.xml.transform.dom",
                "javax.xml.transform.sax",
                "javax.xml.transform.stream",
                "javax.xml.validation",
                "javax.xml.xpath"
                ]

    c = 0
    for pack in packages:
        topcount(apath,pack,acdict,aperclass,aset)
        c += 1
        topcount(javapath,pack,jcdict,jperclass,jset)
        #topcount(gnupath,pack,acdict,aperclass,aset)

    print "%d packages analyzed" % (len(packages))
    print "\nJava Analysis"
    report(jcdict,jperclass)
    print "\nAndroid Analysis"
    report(acdict,aperclass)
    print "\n−−−−−"

    print "common package/private"
    inter = jset&aset
    for name in inter:
        print name

    print "\nAndroid\n−−−−−−"
    for name in aset:
        print name
    print "\nJava\n−−−−−−"
    for name in jset:
        print name



if __name__ == "__main__":
    analyze()
```



```
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''

import os,collections

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aset = set()
apack = {}


jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jset = set()
jpack = {}

public_ids = ["public class",
              "public abstract class",
              "public interface"]

logger = open("classlog","w")


def is_class(line):

    for pub in public_ids:
        if line.startswith(pub):
            return True

    pin = line.find("public")
    cin = line.find("class")
    if pin != −1 and cin != −1 and pin < cin:
        return True
    return False

def do_one(onepath,cset,cpack):
    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package−info.java"):
        return True
    f = open(onepath)

    pcount = 0
    first = True
    public = False
    for line in f:

        line = line.strip()

        if first and is_class(line):
            #print "class",onepath,line
            base = os.path.basename(onepath)
            cset.add(onepath)
            cpack[base] = onepath
            public = True
            break

    return public


def topcount(basepath,cset,cpack):
  '''
  counting public classes and interfaces, use basepath as /java or /javax
  '''

    for top in os.listdir(basepath):
        top_path = os.path.join(basepath,top)
```

```
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)

            print "recurse on ",top_path
            logger.write("recurse on "+top_path+"\n")
            topcount(top_path,cset,cpack)
        else:
            c = do_one(top_path,cset,cpack)
            if not c:
                #print "no public",top_path,top
                pass
            #print "%s has %d public" % (top_path,c)
def revs(s):
    return s[::-1]

def analyze():

    jpath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"

    for toplevel in [""]:
        topcount(jpath+toplevel,jset,jpack)


    names = sorted(jset)
    for i,name in enumerate(names):
        logger.write(str(i)+"\t"+name+"\n")
        print "%d\t%s" % (i,name)

    logger.close()

if __name__ == "__main__":
    analyze()
```



```
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''

import os,collections

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aset = set()
apack = {}


jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jset = set()
jpack = {}

logger = open("packagelog","w")




def topcount(basepath,cset,cpack):
    '''
    looking for package names, find .java file, it's a package
    '''

    jfound = False
    for top in os.listdir(basepath):
        top_path = os.path.join(basepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)

            #print "recurse on ",top_path
            #logger.write("recurse on "+top_path+"\n")
            topcount(top_path,cset,cpack)
        else:
            if top_path.endswith(".java"):
                jfound = True

    if jfound:
        #print ".java found in ",top_path
        cset.add(basepath)


def analyze():

    jpath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    apath = "/Users/ola/expert/google/SOURCE/" #/libcore/luni/src/main/java"

    packdirs = ["frameworks/base/core/java", "libcore/luni/src/main"]

    for dir in packdirs:
        #topcount(jpath,jset,jpack)
        topcount(apath+dir,aset,apack)
        #topcount(apath,pack,aset)


    names = sorted(aset)
    for i,name in enumerate(names):
        logger.write(str(i)+"\t"+name+"\n")
        print "%d\t%s" % (i,name)

    logger.close()

if __name__ == "__main__":
    analyze()
```

```python
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''
import os,collections

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aprivdict = {}
aset = set()
amethnames = []

jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jprivdict = {}
jset = set()
jmethnames = []

gcdict = collections.defaultdict(int)
gperclass = collections.defaultdict(int)
gprivdict = {}
gset = set()
gmethnames = []


afunclist = []
jfunclist = []
gfunclist = []

methnames = []

public_ids = ["public class",
              "public abstract class",
              "public interface",
              "protected class",
              "protected",
              "public"]

def is_func(line):
    if "new " in line:
        return False
    parts = line.split()
    if line.startswith("public") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    if line.startswith("private") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    return False


def getClass(path):
    '''
     path ends with .java, return class name preceding .java including preceding .
     e.g., for java/lang/Arrays, return .Arrays
    '''
    nm = path[:-5]
    index = nm.rfind("/")
    return "."+nm[index+1:]


def do_one(packname,onepath,cdict,perclass,cset,funclist,privdict,methnames):

    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package-info.java"):
        return True
    f = open(onepath)

    class_name = getClass(onepath)

    pcount = 0
```



```python
    first = True
    public = False
    pubf = 0
    privf = 0
    for line in f:

        line = line.strip()

        if is_func(line):
            methnames.append(line)
            if line.startswith("public"):
                pubf += 1
            else:
                privf += 1
                nm = packname+class_name
                if not nm in privdict:
                    privdict[nm] = []
                privdict[nm].append(line)

        if first and line.startswith("class "):
            #print "class",onepath,line
            base = os.path.basename(onepath)
            cset.add(base)

        pfound = False
        for pub in public_ids:

            if line.startswith(pub):
                if first:
                    first = False
                    if line.find("public") >= 0 or line.find("protected") >= 0:
                        public = True
                    else:
                        print "big problem",onepath,pub,line
                if line.find("protected") < 0:
                    pcount += 1
                cdict[pub] += 1
                pfound = True
                if line.find("class") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
                elif line.find("interface") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
                break

    f.close()

    perclass[pcount] += 1
    if pcount == 0:
        #print "%s = %d" % (onepath,pcount)
        pass

    funclist.append((pubf,privf))
    return public

def topcount(basepath,packname,cdict,perclass,cset,funclist,privdict,methnames):
    parts = packname.split(".")
    pathize = '/'.join(parts)
    packagepath = os.path.join(basepath,pathize)
    for top in os.listdir(packagepath):
        top_path = os.path.join(packagepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)
            pass
        else:
            c = do_one(packname,top_path,cdict,perclass,cset,funclist,privdict,m
ethnames)
            if not c:
                #print "no public",top_path,top
                pass
            #print "%s has %d public" % (top_path,c)

def func_stats(coll):
    low = 0
```

```
    word_total = 0
    wt_count = 0
    nonlow = 0
    getter = 0
    setter = 0
    req = 0

    obj_names = ["toString", "hashCode", "notifyAll", "getClass"]

    for nm in coll:
        if nm.islower():
            low += 1
            #print "\t lower",nm
        else:
            wc = 0
            for i,ch in enumerate(nm):
                if ch.isupper() and i > 0 and nm[i-1].islower():
                    wc += 1

            wc += 1
            #word_total += wc
            nonlow += 1

            if nm.startswith("get"):
                getter += 1
            elif nm.startswith("set"):
                setter += 1
            elif nm in obj_names:
                req += 1
            else:
                word_total += wc
                wt_count += 1

    print "total = %d, one = %d more = %d\n" % (nonlow+low,low,nonlow)
    print "perc = %f avg = %f\n" % (1.0*low/(low+nonlow),1.0*word_total/wt_count)
    print "non simple = %d\n" % (wt_count)

    print "getter = %d, setter = %d, req = %d, total = %d\n" % (getter,setter,req,req+getter+se
tter)

def funcalyze(methnames):
    all_names = set()
    names = []
    for meth in methnames:
        if meth.startswith("public"):
            nameEnd = meth.find("(")
            if nameEnd == -1:
                print "error on ",meth
            else:
                name = meth[:nameEnd]
                space = name.rfind(" ")
                mname = name[space+1:]
                all_names.add(mname)
                names.append(mname)

    print "total = %d, unique = %d\n" % (len(names), len(all_names))
    print "unique"
    func_stats(all_names)
    print "total"
    func_stats(names)

    meth_counts = [(names.count(nm),nm) for nm in all_names]
    smc = sorted(meth_counts, reverse=True)
    print "top func occurrences"
    for pair in smc[:20]:
        print pair


    return all_names


def report(cdict,perclass,funclist,privdict,methnames):
```



```
    uset = funcalyze(methnames)


    ctotal = 0
    for key in cdict:
        if key.find("public") < 0:
            continue
        print "%s occurrences = %d" % (key,cdict[key])
        if key.find("class") >= 0 or key.find("interface") >= 0:
            ctotal += cdict[key]
    print "----"
    print "public class/interface total = %d" % (ctotal)

    ctotal = 0
    for key in cdict:
        if key.find("protected") < 0:
            continue
        print "%s occurrences = %d" % (key,cdict[key])
        if key.find("class") >= 0 or key.find("interface") >= 0:
            ctotal += cdict[key]
    print "----"
    print "protected class/interface total = %d" % (ctotal)

    print "per class method counts"
    print "# methods\t#classes"
    total = 0
    levels = collections.defaultdict(int)
    levlist = [0,1,6,11,16,21,51,101,100001]
    for method_count in sorted(perclass.keys()):
        print "%d\t%d" % (method_count,perclass[method_count])
        total += method_count*perclass[method_count]
        for lev in xrange(1,len(levlist)):
            if levlist[lev-1] <= method_count < levlist[lev]:
                levels[lev] += perclass[method_count]
    print "-----"
    print "total methods = %d" % (total)
    print "\n---summary--"
    total = 0
    for lev in xrange(1,len(levlist)):
        print "perclass from %d to %d = %d" % (levlist[lev-1],levlist[lev]-1,levels[le
v])
        total += levels[lev]
    print "total = %d" % (total)

    print "size of funclist = %d" % (len(funclist))
    total = 0
    totalMeths = 0
    totalPriv = 0
    for x in funclist:
        totalMeths += x[0] + x[1]
        totalPriv += x[1]
        if x[0] != 0 or x[1] != 0:
            total += 100.0*x[0]/(x[1]+x[0])
    print "average = %f" % (total/len(funclist))
    print "total meths = %d" % (totalMeths)
    print "total private = %d" % (totalPriv)
    return uset

def analyze():

    apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
    javapath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    gnupath = "/Users/ola/expert/google/source-gnu/classpath-0.98"

    packages = ["java.awt.font",
                "java.beans",
                "java.io",
                "java.lang",
                "java.lang.annotation",
                "java.lang.ref",
                "java.lang.reflect",
                "java.math",
```

```
                    "java.net",
                    "java.nio",
                    "java.nio.channels",
                    "java.nio.channels.spi",
                    "java.nio.charset",
                    "java.nio.charset.spi",
                    "java.security",
                    "java.security.acl",
                    "java.security.cert",
                    "java.security.interfaces",
                    "java.security.spec",
                    "java.sql",
                    "java.text",
                    "java.util",
                    #"java.util.concurrent",
                    #"java.util.concurrrent.atomic",
                    #"java.util.concurrent.locks",
                    "java.util.jar",
                    "java.util.logging",
                    "java.util.prefs",
                    "java.util.regex",
                    "java.util.zip",
                    "javax.crypto",
                    "javax.crypto.interfaces",
                    "javax.crypto.spec",
                    "javax.net",
                    "javax.net.ssl",
                    "javax.security.auth",
                    "javax.security.auth.callback",
                    "javax.security.auth.login",
                    "javax.security.auth.x500",
                    "javax.security.cert",
                    "javax.sql",
                    "javax.xml",
                    "javax.xml.datatype",
                    "javax.xml.namespace",
                    "javax.xml.parsers",
                    "javax.xml.transform",
                    "javax.xml.transform.dom",
                    "javax.xml.transform.sax",
                    "javax.xml.transform.stream",
                    "javax.xml.validation",
                    "javax.xml.xpath"
                    ]

    for pack in packages:
        topcount(javapath,pack,jcdict,jperclass,jset,jfunclist,jprivdict,jmethna
mes)
        topcount(apath,pack,acdict,aperclass,aset,afunclist,aprivdict,amethnames
)
        topcount(gnupath,pack,gcdict,gperclass,gset,gfunclist,gprivdict,gmethnam
es)

    print "%d packages analyzed" % (len(packages))
    print "\nJava Analysis"
    juset = report(jcdict,jperclass,jfunclist,jprivdict,jmethnames)
    print "\nAndroid Analysis"
    auset = report(acdict,aperclass,afunclist,aprivdict,amethnames)
    print "\nGnuClasspath Analysis"
    report(gcdict,gperclass,gfunclist,gprivdict,gmethnames)
    print "\n−−−−−"

    jmset = juset
    amset = auset
    inter = jmset&amset
    aonly = amset−jmset
    jonly = jmset−amset
    print "android only count = ",len(aonly),len(amset)
    print "java only count = ",len(jonly),len(jmset)
    print "android only"
    for i,n in enumerate(sorted(aonly)):
        print i,n
    print "java only"
```



```
    for i,n in enumerate(sorted(jonly)):
        print i,n


    privlog = open("privatelog","w")
    for pack in aprivdict:
        if pack in jprivdict:
            line = "package class private {0!s}\n".format(pack)
            print "package class private %s" % (pack)
            privlog.write(line)
            for priv in aprivdict[pack]:
                line = "\tAndroid {0!s}\n".format(priv)
                privlog.write(line)
                #print "\tAndroid %s" % (priv)
                if priv in jprivdict[pack]:
                    privlog.write("\t\talso in Java\n")
                    #print "\t\talso in Java"
            for priv in jprivdict[pack]:
                if not priv in aprivdict[pack]:
                    privlog.write("\tJava "+priv+"\n")
                    #print "\tJava %s" % (priv)
    privlog.close()




#    print "common package/private"
#    inter = jset&aset
#    for name in inter:
#        print name
#
#    print "\nAndroid\n−−−−−−"
#    for name in aset:
#        print name
#    print "\nJava\n−−−−−−"
#    for name in jset:
#        print name




if __name__ == "__main__":
    analyze()
```

```python
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''
import os,collections


def do_one(packname,onepath):
    '''
    return number of lines in onepath if a .java file
    '''

    if not onepath.endswith(".java"):
        return 0
    if onepath.endswith("package-info.java"):
        return 0

    f = open(onepath)

    lcount = 0
    for line in f:
        lcount += 1

    f.close()
    return lcount

def topcount(basepath,packname):
    parts = packname.split(".")
    pathize = '/'.join(parts)
    packagepath = os.path.join(basepath,pathize)
    total = 0
    ftot = 0
    for top in os.listdir(packagepath):
        top_path = os.path.join(packagepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)
            pass
        else:
            c = do_one(packname,top_path)
            if c != 0:
                ftot += 1
                total += c
    return (total,ftot)

def analyze48():

    apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
    javapath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    gnupath = "/Users/ola/expert/google/source-gnu/classpath-0.98"

    packages = ["java.awt.font",
                "java.beans",
                "java.io",
                "java.lang",
                "java.lang.annotation",
                "java.lang.ref",
                "java.lang.reflect",
                "java.math",
                "java.net",
                "java.nio",
                "java.nio.channels",
                "java.nio.channels.spi",
                "java.nio.charset",
                "java.nio.charset.spi",
                "java.security",
                "java.security.acl",
                "java.security.cert",
                "java.security.interfaces",
                "java.security.spec",
                "java.sql",
```



```python
                "java.text",
                "java.util",
                #"java.util.concurrent",
                #"java.util.concurrrent.atomic",
                #"java.util.concurrent.locks",
                "java.util.jar",
                "java.util.logging",
                "java.util.prefs",
                "java.util.regex",
                "java.util.zip",
                "javax.crypto",
                "javax.crypto.interfaces",
                "javax.crypto.spec",
                "javax.net",
                "javax.net.ssl",
                "javax.security.auth",
                "javax.security.auth.callback",
                "javax.security.auth.login",
                "javax.security.auth.x500",
                "javax.security.cert",
                "javax.sql",
                "javax.xml",
                "javax.xml.datatype",
                "javax.xml.namespace",
                "javax.xml.parsers",
                "javax.xml.transform",
                "javax.xml.transform.dom",
                "javax.xml.transform.sax",
                "javax.xml.transform.stream",
                "javax.xml.validation",
                "javax.xml.xpath"
                ]

    jpair = [0,0]
    apair = [0,0]
    gpair = [0,0]
    for pack in packages:
        j = topcount(javapath,pack)
        a = topcount(apath,pack)
        g = topcount(gnupath,pack)
        jpair[0] += j[0]
        jpair[1] += j[1]
        apair[0] += a[0]
        apair[1] += a[1]
        gpair[0] += g[0]
        gpair[1] += g[1]



    print "Java = %d %d \nAndroid = %d %d\nGnu = %d %d\n" % (jpair[0],jpair[1],apair[0],a
pair[1],gpair[0],gpair[1])


if __name__ == "__main__":
    analyze48()
```

```python
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''
import os,collections

#global vars aren't used as global, but passed as parameters
#in different calls

#android vars

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aset = set()
apack = {}


#java vars

jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jset = set()
jpack = {}

logger = open("sloclog","w")

idents = ["AclEntryImpl.java",
          "AclImpl.java",
          "GroupImpl.java",
          "OwnerImpl.java",
          "PermissionImpl.java",
          "PrincipleImpl.java",
          "AclEnumerator.java",
          "PolicyNodeImpl.java",
          "CodeSourceTest.java",
          "CollectionCertStoreParametersTest.java",
          "TimSort.java",
          "ComparableTimSort.java"]


def do_one(onepath,cset,cpack):

    if onepath.endswith("package-info.java"):
        return 0

    endings = [".java", ".h", ".c", ".cpp"]
    ok = False
    for e in endings:
        if onepath.endswith(e):
            ok = True
    if not ok:
        return 0

    # for test files, uncomment below, find files with /test/ in path
#    dex = onepath.rfind("/")
#    if dex == -1:
#        print "trouble on ", onepath
#    base = onepath[:dex]
#    lst = base[base.rfind("/")+1:]
#    if lst != "test":
#        return 0
#    print onepath


    # for 12 files at issue in idents uncomment code
#    ok = False
#    for ids in idents:
#        if onepath.endswith(ids):
#            ok = True
#    if not ok:
```



```python
#        return 0
#
#    print onepath

    f = open(onepath)
    pcount = 0
    for line in f:

        pcount += 1


    #print pcount,onepath
    return pcount



def topcount(basepath,cset,cpack):
    '''
    traverse directory structure looking for files and data via do_one
    '''

    fcount = 0
    lcount = 0


    for top in os.listdir(basepath):
        top_path = os.path.join(basepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)

            #print "recurse on ",top_path
            #logger.write("recurse on "+top_path+"\n")
            res = topcount(top_path,cset,cpack)
            fcount += res[0]
            lcount += res[1]
        else:
            c = do_one(top_path,cset,cpack)
            if c != 0:
                lcount += c
                fcount += 1

    return (fcount,lcount)


def analyze():

    jpath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    apath = "/Users/ola/expert/google/SOURCE/"

    #dirs = ["libcore", "frameworks/base/core"] #/luni/src/main/java"]
    dirs = [""]
    lcount = 0
    fcount = 0
    for dir in dirs:
        c = topcount(apath+dir,jset,jpack)
        fcount += c[0]
        lcount += c[1]

    print "files = %d, lines = %d\n" % (fcount,lcount)

if __name__ == "__main__":
    analyze()
```